



National Aeronautics and  
Space Administration

NASA CR-188246

Lyndon B. Johnson Space Center  
Houston, Texas 77058

**SPACE GENERIC OPEN AVIONICS ARCHITECTURE  
(SGOAA)  
REFERENCE MODEL TECHNICAL GUIDE**

April 1993

Richard B. Wray  
John R. Stovall

(This revision supersedes LESC-30347, issued December 1992)

Prepared by:

Lockheed Engineering & Sciences Company  
Houston, Texas

Job Order 60-911  
Contract NAS 9-17900

for

FLIGHT DATA SYSTEMS DIVISION  
JOHNSON SPACE CENTER

LESC-30347-A

N93-31032

Unclas

G3/19 0176818

(NASA-CR-188246) SPACE GENERIC  
OPEN AVIONICS ARCHITECTURE (SGOAA)  
REFERENCE MODEL TECHNICAL GUIDE  
(Lockheed Engineering and Sciences  
Co.) 195 p




**SPACE GENERIC OPEN AVIONICS ARCHITECTURE  
(SGOAA)  
REFERENCE MODEL TECHNICAL GUIDE**

April 1993

Richard B. Wray, Advanced Systems Engineering Specialist  
John R. Stovall, Advanced Systems Engineering Specialist

APPROVED BY:

  
G.L. Clouette, Project Integration Specialist  
Flight Data Systems Department

  
D.M. Pruett, Manager, Advanced Programs  
Flight Data Systems Division

Prepared by:

Lockheed Engineering & Sciences Company  
Houston, Texas

Job Order 60-911  
Contract NAS 9-17900

for

FLIGHT DATA SYSTEMS DIVISION  
JOHNSON SPACE CENTER

LESC-30347-A





## DOCUMENT CHANGE RECORD

The following table summarizes the change activity associated with this document.

ISSUE AND DATE	CHANGE SUMMARY	SECTION

## **PREFACE**

This document has been produced by Mr. Richard B. Wray and Mr. John R. Stovall of Lockheed Engineering and Sciences Company (LESC), the codevelopers of the avionics architectures and standards represented in this document. The contributions of Mr. Ben Doeckel of LESG who participated in early development of the concepts for the avionics architectures and standards represented in this document is acknowledged. Special acknowledgement is also given to Mr. Dave Pruett of the Johnson Space Center for his support of the Advanced Architecture Analysis, assistance in the development of the avionics architecture and constructive criticisms of the proposed standard. Major inputs for Section 2.2 and Appendix B on Architecture Requirements were provided by Mr. Rich Flanagan and Mr. Art Van Ausdal of the Boeing Defense and Space Group.

# CONTENTS

Section	Page
1. INTRODUCTION .....	1-1
1.1 <u>OBJECTIVE</u> .....	1-4
1.2 <u>PURPOSE</u> .....	1-4
1.3 <u>SCOPE</u> .....	1-4
1.4 <u>APPROACH</u> .....	1-4
1.5 <u>ORGANIZATION OF DOCUMENT</u> .....	1-7
2. SPACE GENERIC OPEN AVIONICS ARCHITECTURE REFERENCE REQUIREMENTS AND MODELS.....	21-1
2.1 <u>SPACE GENERIC OPEN AVIONICS ARCHITECTURE         BACKGROUND</u> .....	21-3
2.1.1 MODEL DEVELOPMENT PROCESS.....	21-4
2.1.1.1 <u>Development Background</u> .....	21-4
2.1.1.2 <u>Architectural Principles Required</u> .....	21-4
2.1.1.3 <u>Architecture Assumptions and Ground Rules</u> .....	21-5
2.1.2 REFERENCE MODEL DEFINITIONS.....	21-9
2.1.3 RELATED ARCHITECTURAL STANDARDS AND REFERENCE MODELS FOR AVIONICS .....	21-11
2.1.3.1 <u>POSIX Open System Environment</u> .....	21-11
2.1.3.2 <u>OSI Model</u> .....	21-13
2.2 <u>SPACE GENERIC OPEN AVIONICS ARCHITECTURE         REQUIREMENTS</u> .....	22-1
2.2.1 REQUIREMENTS OVERVIEW.....	22-1
2.2.1.1 <u>Open Systems Requirements</u> .....	22-1
2.2.1.2 <u>Lower Level Standard Selection</u> .....	22-2
2.2.2 ARCHITECTURE FEATURES.....	22-4
2.2.2.1 <u>Requirements Architecture</u> .....	22-4

<b>Section</b>	<b>Page</b>
2.2.2.2 <u>Critical Interfaces</u> .....	22-4
2.2.2.3 <u>Service Interfaces</u> .....	22-4
2.2.2.4 <u>Resource Control</u> .....	22-4
2.2.2.5 <u>Commonality</u> .....	22-5
2.2.2.6 <u>Interface Standardization</u> .....	22-5
2.2.2.7 <u>Crew Override</u> .....	22-5
2.2.2.8 <u>Dependability Management</u> .....	22-6
2.2.2.9 <u>Data System Services</u> .....	22-6
2.2.2.10 <u>Growth and Spare Capacity</u> .....	22-7
2.2.2.11 <u>Modularity</u> .....	22-7
2.2.2.12 <u>Service Transparency</u> .....	22-7
2.2.2.13 <u>Technology Transparency</u> .....	22-7
2.2.2.14 <u>Interoperability</u> .....	22-7
2.2.2.15 <u>Goals</u> .....	22-7
2.2.3 EXTERNAL INTERFACES .....	22-10
2.2.3.1 <u>External Interface Requirements</u> .....	22-10
2.2.3.2 <u>SGOAA Development Interface Definition Objectives</u> .....	22-11
2.2.4 SGOAA DEVELOPMENT FUNCTIONAL REQUIREMENTS.....	22-13
2.2.4.1 <u>Process and Data Requirements</u> .....	22-14
2.2.4.2 <u>Performance and Quality Engineering Considerations</u> .....	22-18
23 <u>SPACE GENERIC OPEN AVIONICS ARCHITECTURE DETAILED REQUIREMENTS DESCRIPTION</u> .....	23-1
2.3.1 <u>GENERIC SYSTEM ARCHITECTURE REQUIREMENTS DESCRIPTION</u> .....	23-2
2.3.2 <u>ARCHITECTURE INTERFACE MODEL REQUIREMENTS DESCRIPTION</u> .....	23-4

Section	Page
2.3.2.1	<u>Class 1 - Hardware-to-Hardware Direct Interfaces</u> .....23-4
2.3.2.2	<u>Class 2 - Hardware-to-System Software Direct Interfaces</u> ...23-15
2.3.2.3	<u>Class 3 - System Software-to-Software (Local) Direct Interfaces</u> .....23-17
2.3.2.4	<u>Class 4 - System Software-to-System Software Logical Interfaces</u> ..... 23-21
2.3.2.5	<u>Class 5 - System Software-to-Applications Software (Local) Direct Interfaces</u> .....23-24
2.3.2.6	<u>Class 6 - Applications Software-to-Applications Software Logical Interfaces</u> .....23-27
24	<u>SGOAA RELATIONSHIPS TO POSIX</u> .....24-1
24.1	POSIX OVERVIEW .....24-1
24.1.1	<u>Application Platform</u> .....24-1
24.1.2	<u>Application Program Interface (API)</u> .....24-2
24.1.3	<u>External Environment</u> .....24-5
24.1.4	<u>External Environment Interface (EEI)</u> .....24-5
24.2	SGOAA INTERFACE CLASS RELATIONSHIPS TO POSIX .....24-7
24.2.1	<u>Class 1 Hardware-to-Hardware Interfaces (DIRECT)</u> .....24-7
24.2.2	<u>Class 2 Hardware-to-System Software Interface (DIRECT)</u> .....24-8
24.2.3	<u>Class 3 System Software-to-Software (Local DIRECT)</u> .....24-9
24.2.4	<u>Class 4 System Software-to-System Software Interfaces (LOGICAL)</u> .....24-9
24.2.5	<u>System Software-to-Application Software Class 5 (DIRECT)</u> .....24-10
24.2.6	<u>Class 6 Application Software-to-Application Software (LOGICAL)</u> .....24-10

<b>Section</b>	<b>Page</b>
3. THE SPACE GENERIC OPEN AVIONICS ARCHITECTURE APPLIED.....	3.1-1
3.1 <u>POTENTIAL SPACE GENERIC AVIONICS FUNCTIONS</u> .....	3.1-2
3.2 <u>DATA AND CONTROL FLOW DIAGRAM CONVENTIONS</u> .....	3.2-1
3.3 <u>FUNCTIONAL ARCHITECTURES</u> .....	3.3-1
3.4 <u>KEY INTEGRATING SOFTWARE SUBSYSTEM ARCHITECTURES</u> .....	3.4-1
3.4.1 SPACE DATA SYSTEM SERVICES ARCHITECTURE.....	3.4-1
3.4.1.1 <u>SDSS Control Modes</u> .....	3.4-4
3.4.1.2 <u>Standard Data Services Manager</u> .....	3.4-5
3.4.1.3 <u>Network Services Manager</u> .....	3.4-7
3.4.1.4 <u>Data System Manager</u> .....	3.4-11
3.4.1.5 <u>Operating System</u> .....	3.4-15
3.4.1.6 <u>Data Base Manager</u> .....	3.4-17
3.4.2 SPACE OPERATIONS CONTROL SUBSYSTEM.....	3.4-19
3.4.2.1 <u>Vehicle Controller</u> .....	3.4-19
3.4.2.2 <u>Command Controller</u> .....	3.4-21
3.4.2.3 <u>Systems Controller</u> .....	3.4-21
3.4.2.4 <u>Communications &amp; Tracking Control</u> .....	3.4-23
3.4.2.5 <u>Crew Manager</u> .....	3.4-23
3.4.2.6 <u>Integrated Logistics Control</u> .....	3.4-23
3.4.2.7 <u>Payload and Science Operations Control</u> .....	3.4-24

<b>Section</b>	<b>Page</b>
3.5 <u>SPACE DATA SYSTEM ARCHITECTURE APPLICATION</u> .....	3.5-1
3.5.1 SPACE STATION APPLICATION .....	3.5-1
3.5.2 COMMON LUNAR LANDER APPLICATION .....	3.5-3
3.5.2.1 <u>Space Operations Control Subsystem Requirements Tailoring</u> .....	3.5-3
3.5.2.2 <u>Space Data System Services Requirements Tailoring</u> .....	3.5-5
3.5.2.3 <u>Space Data System Hardware Requirements Tailoring</u> .....	3.5-5
3.5.2.4 <u>GAP Internal Architecture Requirements Tailoring</u> .....	3.5-7
3.5.2.5 <u>Lessons Learned</u> .....	3.5-9
4. CONCLUSION AND RECOMMENDATIONS.....	4-1
4.1 <u>CONCLUSIONS</u> .....	4-1
4.2 <u>RECOMMENDATIONS</u> .....	4-2

<b>APPENDICES</b>	<b>Page</b>
A. SGOAA DEFINITIONS .....	A-1
B. ARCHITECTURE SPECIFICATION TABLES .....	B-1
C. ARCHITECTURE REVIEW COMMENTS AND RESPONSES .....	C-1
D. LIST OF REFERENCES.....	D-1

## TABLES

<b>Table</b>	<b>Page</b>
2-1 Critical Condition Categories .....	2.2-9
2-2 Partial List of Related Standards.....	2.2-11
2-3 Message Transfer Service Grade .....	2.2-14
2-4 Message Transfer Service Grade Characteristics .....	2.2-14
2-5 Capacity Margins for Growth.....	2.2-18
2-6 Reliability Requirements.....	2.2-20
2-7 Availability Requirements.....	2.2-21
2-8 Architectural Interface Classes .....	2.3-5
2-9 SGOAA Relationships to POSIX.....	2.4-7
B-1 SATWG Avionics Architecture Vehicle Study Class Definition.....	B-3
B-2 SATWG FDS Architecture Mission Set Requirements.....	B-4
B-3 SATWG FDS Architecture Service and Interface Requirements.....	B-6
B-4 Sensor Interface.....	B-8
B-5 Effector Interfaces.....	B-8
B-6 Bit Error Rate Characteristics .....	B-8
B-7 Priority Message Transfer Latency (Maximum Time).....	B-9
B-8 Processing Throughput Capacity .....	B-9
B-9 Network Throughput Capacity.....	B-9
B-10 Allocated Memory Space.....	B-10
B-11 Timing Requirements .....	B-10



## FIGURES

Figure	Page
1-1 Mission and Operational Requirements for all Vehicles.....	1-6
2-1 Architecture Reference Model Relationships.....	2.1-2
2-2 Architecture Reference Model Interfaces .....	2.1-2
2-3 One View of the Space Generic Architecture with the assumed processing boundary ...	2.1-3
2-4 Potential Process Partitioning included by the Space Generic Architecture.....	2.1-8
2-5 Hardware Architecture Assumed for the Space Generic Avionics .....	2.1-10
2-6 Open System Environment Model of Applications and Interfaces.....	2.1-11
2-7 SGOAA Functional Interfaces.....	2.2-3
2-8 SGOAA Functional Requirements.....	2.2-9
2-9 Logical System Requirements Flowdown to Direct Design Requirements .....	2.3-1
2-10 System Architecture Model.....	2.3-3
2-11 Architecture Interface Model Interface Class Relationships .....	2.3-6
2-12 Class 1 Hardware to Hardware Direct Interfaces .....	2.3-8
2-13 GAP Hardware to Hardware Interface Standards .....	2.3-10
2-14 Generic Processing External Hardware Architecture and Interfaces for a Space Generic Open Avionics Architecture.....	2.3-11
2-15 POSIX Open System Environment Interfaces Applied to a Station Example of a Standard Hardware Architecture .....	2.3-13
2-16 Generic Processing Internal Hardware Architecture Model.....	2.3-16
2-17 Class 2 Hardware-to-System Software Direct.....	2.3-18
2-18 GAP to Operating System Hardware Drivers .....	2.3-19
2-19 Class 3 System Software-to-Software Direct Interfaces .....	2.3-20
2-20 Operating System Interfaces .....	2.3-22
2-21 Class 4 System Software-to-System Software Logical Interfaces.....	2.3-23
2-22 SDSS Services to Other or Remote Services.....	2.3-25
2-23 Class 5 System Software-to-Applications Software Direct Interfaces.....	2.3-26
2-24 Services to Applications Interfaces.....	2.3-28

<b>Figure</b>	<b>Page</b>
2-25 Class 6 Applications Software-to-Applications Software Logical Interfaces .....	2.3-29
2-26 Class 6 System A Software-to-System B Software Logical Interfaces .....	2.3-31
3-1 Space Generic Avionics Potential Functions Checklist.....	3.1-4
3-2 Space Generic Avionics Core Functional Architecture.....	3.3-1
3-3 Distributed Generic Space Data System Interface Diagram.....	3.3-4
3-4 Space Data System Services .....	3.4-3
3-5 Space Data System Services Provides Multiple Control Modes .....	3.4-4
3-6 Standard Data Services Manager.....	3.4-6
3-7 Network Services Manager.....	3.4-8
3-8 Stack Software and Hardware Partitioning.....	3.4-9
3-9 Network Service Controller.....	3.4-10
3-10 Data System Manager.....	3.4-12
3-11 Operating System Services.....	3.4-15
3-12 Data Base Manager.....	3.4-18
3-13 Space Operations Control Subsystems.....	3.4-20
3-14 Subsystem Coordination through Vehicle Control .....	3.4-21
3-15 Needed Functions in Operations Command Control.....	3.4-22
3-16 Systems Control Optimizes Functionality.....	3.4-23
3-17 Integrated Logistics Control Supplies and Fixes Broken Spacecraft.....	3.4-24
3-18 Payload and Science Operations Control.....	3.4-26
3-19 Generic Avionics Architecture vs. Space Station Design.....	3.5-2
3-20 Space Operations Control Subsystem Requirements Tailoring.....	3.5-4
3-21 Space Data System Services Requirements Tailoring.....	3.5-6
3-22 Space Data System Hardware Requirements Tailoring.....	3.5-7
3-23 GAP Internal Architecture Hardware Requirements Tailoring .....	3.5-8
3-24 Common Lunar Lander Tailored Architecture.....	3.5-10

## **ACRONYMS**

<b>AFE</b>	<b>Aeroassist Flight Experiment</b>
<b>AIM</b>	<b>Architecture Interface Model</b>
<b>ANSI</b>	<b>American National Standards Institute</b>
<b>AP</b>	<b>Application Platform</b>
<b>API</b>	<b>Applications Program Interface</b>
<b>AS</b>	<b>Application Software</b>
<b>BIT</b>	<b>Built-In-Test</b>
<b>BITE</b>	<b>Built-In Test Equipment</b>
<b>C&amp;T</b>	<b>Communicaitons and Tracking</b>
<b>CLL</b>	<b>Common Lunar Lander</b>
<b>D&amp;C</b>	<b>Displays and Controls</b>
<b>DA</b>	<b>Data Acquisition</b>
<b>DD</b>	<b>Data Distribution</b>
<b>DMS</b>	<b>Data Management System</b>
<b>DSM</b>	<b>Data System Manager</b>
<b>ECCV</b>	<b>Earth Crew Capture Vehicle</b>
<b>EE</b>	<b>End Environment</b>
<b>EEI</b>	<b>External Environment Interface</b>
<b>EP</b>	<b>Effector Processing</b>
<b>EP</b>	<b>Embedded Processor</b>
<b>EVA</b>	<b>Extra Vehicular Activity</b>
<b>FB</b>	<b>Future Bus</b>
<b>FDDI</b>	<b>Fiber Distributed Data Interface</b>
<b>FDIR</b>	<b>Fault Detection, Isolation and Control</b>
<b>FDSD</b>	<b>Flight Data Systems Division</b>
<b>GAP</b>	<b>General Avionics Processors</b>
<b>GN&amp;C</b>	<b>Guidance, Navigation and Control</b>
<b>H&amp;S</b>	<b>Health and Status</b>
<b>I/O</b>	<b>Input/Output</b>
<b>IEC</b>	<b>International Electrotechnical Commission</b>
<b>IEEE</b>	<b>Institute of Electrical and Electronic Engineers</b>
<b>IP</b>	<b>Internet Protocol</b>
<b>ISA</b>	<b>Instruction Set Architecture</b>
<b>ISO</b>	<b>International Standards Organization</b>

IVA	Intra Vehicular Activity
JSC	Johnson Space Center
LESC	Lockheed Engineering & Sciences Company
MCMT	Mean Corrective Maintenance Time
MDM	Multiplex-DeMultiplex
MSU	Mass Storage Unit
MTBCF	Mean Time Between Critical Failure
MTBF	Mean Time Between Failure
NASA	National Aeronautics and Space Administration
NC	Non-Critical
NM	Network Manager
NSC	Network Service Controller
NSM	Network Services Manager
NSMID	NSM Interface Definition
OOA	Object Oriented Analysis
OS	Operating System
OSE	Open System Environment
OSI	Open System Interconnect
POSIX	Portable Operating System Interface
RODB	Runtime Object Data Bases
ROM	Remote Operations Manager
RTE	Run Time Environment
SAAP	Space Avionics Architecture Panel
SAE	Society of Automotive Engineers
SAP	Special Avionics Processor
SATWG	Strategic Avionics Technology Working Groups
SC	Safetu Critical
SDP	Standard Data Processor
SDSS	Space Data System Services
SGA	Space Generic Avionics
SGOAA	Space Generic Open Avionics Architecture
SOCS	Space Operations Control Subsystem
TBD	To Be Determined
TCP	Transmission Control Protocol

## **1. INTRODUCTION**

As the United States of America ventures into a new era of Space Exploration with the return to the Moon and then on to Mars and beyond with both manned and unmanned spacecraft, more effective approaches to developing these new spacecraft must be found. These approaches must address cost, schedule and technical performance.

Previous space programs have usually relied upon one major space vehicle being in development or flight at a time. In the future, multiple vehicles such as the Artemis Common Lunar Lander, Manned Lunar Lander, National AeroSpace Plane, Unmanned Mars Lander and even a manned Mars Lander will be in various stages of the development life cycle simultaneously. In order to be able to afford the development of these vehicles, new and evolutionary approaches to the design of these vehicles must be developed. Avionics systems are a prime candidate for the development of evolutionary approaches as there is much commonality between the functions that must be provided for all space vehicles.

This report presents a full description of the Space Generic Open Avionics Architecture (SGOAA) established in [WRA93]. The SGOAA consists of a generic system architecture for the entities in spacecraft avionics, a generic processing architecture and a six class model of interfaces in a hardware/software system. The purpose of the SGOAA is to provide an umbrella set of requirements for applying the generic architecture interface model to the design of specific avionics hardware/software systems. The SGOAA defines a generic set of system interface points to facilitate identification of critical interfaces and establishes the requirements for applying appropriate low level detailed implementation standards to those interfaces points. The generic core avionics system and processing architecture models provided herein are robustly tailorable to specific system applications and provide a platform upon which the interface model is to be applied.

The SGOAA is intended to be used by both avionics system designers and avionics system implementors in the development of open systems architectures for avionics. The system under design shall be expressed in the context of the System Architecture and Generic Processing Architecture as described in Sections 2.3.1 and 2.3.2 respectively. The Architecture Interface Model shall be directly applied to identify the specific interfaces requiring application of lower level standards. The selection of specific lower level

standards is dependent upon unique system requirements, but shall be conducted in accordance with the guidelines described in Section 2.2.1.2.

An open systems architecture can provide the following benefits to future space programs:

- Provide the basis for establishing a set of specifications, standards and procedures that will become common to all systems used in simultaneously operational missions, e.g., to simplify interfaces between multiple vehicles (such as the shuttle and station) when performing a joint mission such as docking.
- Ensure that future avionics systems can be upgraded and maintained with minimal redesign impact to the existing avionics system by establishing the interfaces required to enable modular replacement of hardware and software.
- Promote availability of multiple sources of needed avionics software and hardware by defining standard interfaces.
- Provide a pool of hardware and software modules for multiple program re-use by defining standard interfaces and promoting hardware and software reuse and commonality.
- Insure access to the architecture and its design documentation for any vendor or agency desiring to propose new uses and applications, and to facilitate competition to contain cost growth.

An Advanced Architecture Analysis was conducted by Lockheed Engineering & Sciences Company (LESC) for the National Aeronautics and Space Administration (NASA) Johnson Space Center (JSC) to develop a generic methodology for defining avionics architecture that can be tailored to match the varying requirements of all space vehicles without requiring the system engineering team for each new vehicle to reinvent the requirements analysis and design process. The methodology is presented in reference [WRA91].

An Architecture Requirements Study was conducted for JSC by Boeing Defense and Space Group [BOE91] to develop the requirements for generic software and hardware architecture for space missions. The requirements developed drive the architecture and are incorporated in this document. [BOE91] also presented specific performance parameters for many of these requirements. The performance parameters are considered design

requirements and as such are outside the scope of the SGOAA, but are summarized in Appendix B for reference.

SATWG system design goals and objectives were considered in establishing the following SGOAA design goals:

- (1) Minimize life-cycle costs
- (2) Provide a robust design
- (3) Provide technology transparency

The SGOAA presented in this document is a product of the ongoing Advanced Architecture Analysis task for JSC. The SGOAA consists to date of the following models:

- **A Generic System Architecture Model**
- **A Six Class Architecture Interface Model**
- **A Generic Processing External Hardware Architecture Model**
- **A Generic Processing Internal Hardware Architecture Model**

The SGOAA was developed by first designing the following two functional avionics architectures as described in Section 2.1:

- **Space Data System Services (SDSS) Subsystem Architecture**
- **Space Operations Control Subsystem (SOCS) Architecture**

These are the two key integrating subsystems of the generic system architecture. These architecture are reuse architecture and can be tailored to match the varying requirements of multiple space vehicles in accordance with the goal of the Strategic Avionics Technology Working Groups (SATWG) Space Avionics Architecture Panel (SAAP) to develop families of avionics systems suitable for multiple program use.

The SGOAA and the two functional subsystem architecture presented in this report satisfy the Portable Operating System Interface for Computer Environments (POSIX) "Avionics Software Open System Environment Reference Model" in reference [POSIX91] as discussed in Section 2.1.3.1. The relationships of the SGOAA to the OSI model are discussed in paragraph 2.1.3.2.

## **1.1 OBJECTIVE**

A standard or set of standards is needed for advanced architectures in future space programs. The LESC Advanced Architecture Analysis is being performed under the auspices of D. Pruett, Manager for Advanced Programs for the NASA JSC, Flight Data Systems Division (FDSD). Its objective is to develop a family of architectures for space avionics systems that can be used in any future space vehicle or facility. The information in this document is intended to support the SAAP goal of minimizing life cycle costs, including sustaining engineering costs, of space avionics hardware and software.

## **1.2 PURPOSE**

This document is a technical guide to the proposed SGOAA Standard, and has been produced by LESC. The proposed SGOAA Standard is contained in a separate document [WRA93]. This technical guide presents the results to date of the Advanced Architecture Analysis task to assist the SATWG and SAAP with strategic planning for avionics development in future space programs. All of the work presented has been done at JSC. It is hoped that this report will stimulate participation and contributions from the other NASA centers or SATWG members. Feedback and response from the readers are solicited and should be directed to D. Pruett, NASA Johnson Space Center, EK11, Houston, TX 77058; NASAMAIL: dpruett; INTERNET: dpruett@asd1.jsc.nasa.gov; or to R. Wray or J. Stovall, LESC, 2400 NASA Road 1, Mail Code C18, Houston, TX 77058. Future revisions of this document will be published as the study task progresses.

## **1.3 SCOPE**

The results presented here are the third release of the results of a continuing effort by JSC and LESC to develop an overall generic architecture for avionics, which can be applied to the development of all future space mission avionics systems.

## **1.4 APPROACH**

The approach taken by this study team was to gather data on existing space programs (Space Shuttle and Space Station) in order to develop a comprehensive understanding of the functions and services that space avionics systems must provide in order to develop an architecture that would not be a "blue sky" approach, but would be based on reality.



The architecture development effort also monitored efforts by the SATWG and its contractors to build generic, standard or open architecture. Of special note was the incorporation of the avionics software Open System Environment Reference Model, as presented in reference [POSIX91], into this architecture.

The main source of the underlying requirements for the SGOAA (outside of shuttle and station documentation) were References [GD90A] and [BOE91] from the SATWG. In particular, the driving requirement was for an open architecture allowing multi-vendor sources for components, interchangeable and interoperable elements with reduced complexity and cost and common elements. This led to the approach of using common space applications software relying on common data system services. The requirement for a robust modular system led to the need to avoid preconceptions on partitioning functions between subsystems, in order to create a modular structure which facilitates hardware and software reuse on multiple programs/missions and modular upgrades. Upgrades may occur due to technology insertion, maintenance actions or changing mission requirements. The requirement to support technology upgrades led to a structure which isolated specific technologies from requirements implementation to the extent possible.

As shown in Figure 1-1, a starting requirement for the architecture development was that it should be adaptable to all future space missions including:

- Surface-to-orbit missions to reach low Earth orbit, Lunar orbit or Mars orbit from the local planetary surface.
- Docking and berthing operations between adjacent space platforms such as shuttles arriving at the space station, heavy lift vehicles linking with orbiting vehicles, Lunar ascent vehicles mating with their orbiting transfer vehicles, etc.
- Orbit station keeping operations by orbiting platforms maintaining stable orbits around a planetary body, such as the space station in Earth orbit, the Lunar transfer vehicle on Moon orbit, or the Mars space station in Mars orbit.
- Orbit-to-orbit transfers, which may be from low to high Earth orbit, from Earth to the Moon, or from Earth (or another planet) into deep space.
- Orbit-to-surface missions to land on the Earth, Moon or Mars from an orbiting or arriving space vehicle.

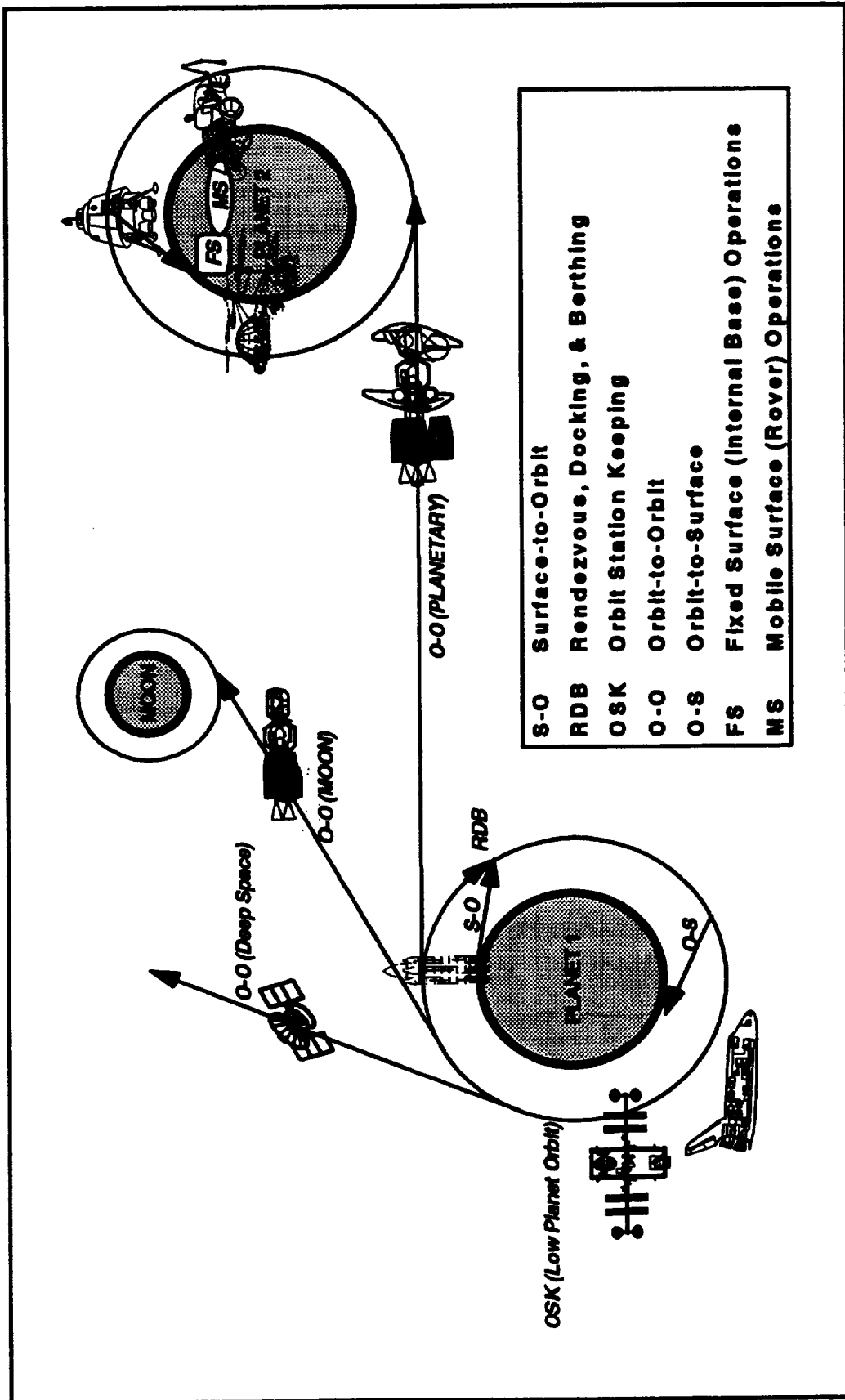


Figure 1-1. Mission and Operational Requirements for all Vehicles

- Fixed surface operations in a fixed base such as on the Moon base or Mars base, where such a base may be performing permanently manned complex operations or just temporarily manned exploration operations.
- Mobile surface operations in rover or similar vehicles moving on the planet surface.

The SGOAA incorporates both hardware and software into one architecture. Since all space flight data systems are heavily dependent upon software, this was a primary consideration in ensuring effective software requirements and effective software-to-software and software-to-hardware interface definition.

## **1.5 ORGANIZATION OF DOCUMENT**

Section 2 describes the SGOAA background requirements (based on work by Boeing in reference [BOE91]), and the SGOAA itself, including the Generic System Architecture Model in paragraph 2.3.1, the Architecture Interface Model in paragraph 2.3.2 and the Generic Processing Hardware Architecture Model in paragraphs 2.3.2.1.1, 2.3.2.1.2 and 2.3.2.1.3.. Section 3 discusses development of the detailed Space Generic Avionics functional architectures (SDSS and SOCS). It also describes the results in applying the generic models to two space programs: the Space Station and the Common Lunar Lander. Section 4 discusses the conclusions reached to the present and the recommendations for continuing development.



## **2. SPACE GENERIC OPEN AVIONICS ARCHITECTURE REFERENCE REQUIREMENTS AND MODELS**

Standards are needed in the development of generic open avionics architecture, as previously noted. But standards without a reference model are very difficult to apply, and hence of limited utility. A reference model is needed to show how to apply the standards, to check if the application is complete and consistent, and to identify the effort remaining for completion of an application. The reference model provides the structure on which the standard is built, and from which applications using the structure can be developed. Section 2.1 provides the background, processes used and ground rules for developing the model. Section 2.2 defines the architecture requirements. The reference models are described in Section 2.3: The SGOAA Generic System Architecture Model in Section 2.3.1, the Architecture Interface Model (AIM) in Section 2.3.2, the Hardware Interface Architecture Model in Section 2.3.2.1.1, the Generic Processing External Hardware Architecture Model in Section 2.3.2.1.2, and the Generic Processing Internal Hardware Architecture Model in Section 2.3.2.1.3. Detailed relationships between the POSIX and SGOAA models are described in Section 2.4. Architecture applications of the reference model are described in Section 3.

The POSIX Open System Environment (OSE) Reference Model is the basis for incorporating application software portability and interoperability into the five models that comprise the SGOAA. The POSIX Model can be related to the OSI Model and the SGOAA Interface Model as shown in Figure 2-1. The OSE Model communications links are defined in detail by the Open System Interconnect (OSI) Model for peer-to-peer communications. The OSE Model interface classes are defined in detail by the proposed SGOAA Interface Model. Model relationships are discussed in more detail in Section 2.1.3.

The application of the SGOAA classes to the POSIX OSE model entities and interfaces is shown in Figure 2-2.

The reference model must show how independence is achieved between interface classes in order to treat software in each interface class as a black box with standardized interface specifications. Six interface classes have been developed to achieve independence between each class, to distinguish between logical and physical implementation issues and to separate hardware and software issues. These classes also facilitate the partitioning of software applications which serve to satisfy user requirements and software services which serve to satisfy implementation design requirements.

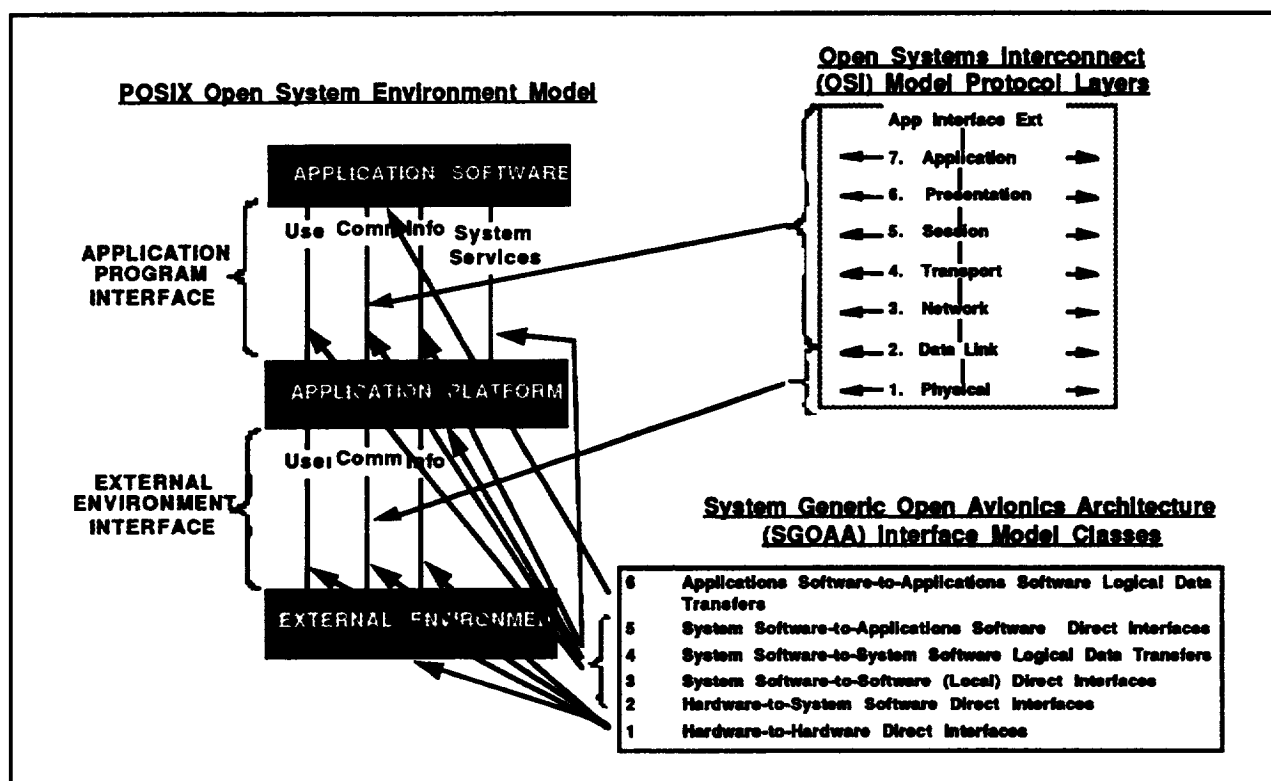


Figure 2-1. Architecture Reference Model Relationships

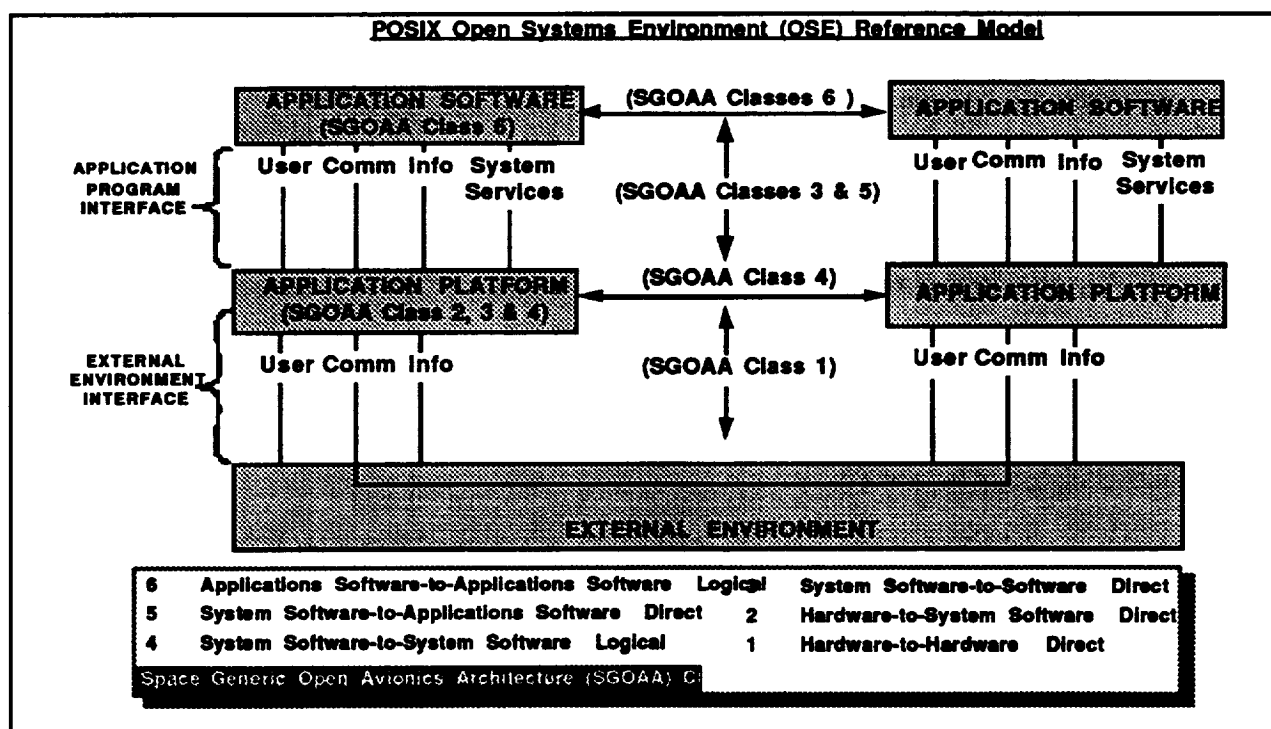


Figure 2-2. Architecture Reference Model Interfaces

## 2.1 SPACE GENERIC OPEN AVIONICS ARCHITECTURE BACKGROUND

A Space Generic Avionics (SGA) Core Functional Architecture was developed as the basis for developing the SGOAA and is shown in Figure 2-3. For this development, an avionics definition was assumed such that the control subsystems for each of the more traditional subsystems (such as Guidance, Navigation and Control (GNC) or Communications and Tracking (C&T)) were within the avionics boundary while the hardware sensors and effectors were outside the avionics boundary to facilitate boundary definition with its attendant conditions and to enable a stronger focus on architecture development.

A key study focus was to define the architecture interface requirements for the functional services needed to enable applications subsystems to operate effectively, i.e., to define the support avionics architectural elements. These are shown by the darkened lines on the operations control application and the data system services bubbles and interfaces in Figure 2-3. This focus provided not only an architectural target, but also some value-added avionics functional definition for operations control and data systems services. (This diagram is not intended to suggest that these are the only interfaces of concern in a space avionics system, nor that the subsystems revolve around the operations control subsystem as a central point of control.)

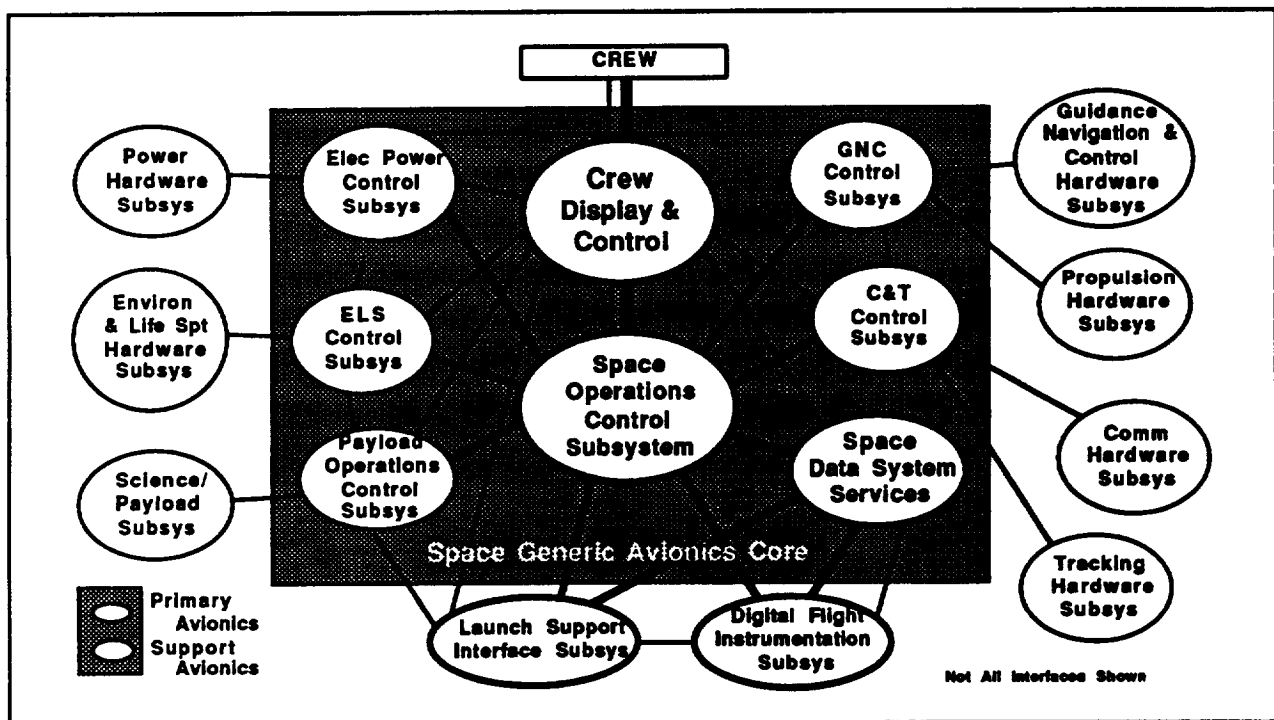


Figure 2-3. One View of the Space Generic Architecture with the assumed processing boundary

Development of the SGA Core functional architecture models are described in the following paragraphs.

### **2.1.1 MODEL DEVELOPMENT PROCESS**

The architectural model for the SGA core architecture was based on the need to provide a reference model which incorporated both hardware and software architectural elements.

*This architecture is not a completed product, rather it is a living architecture standard which can continue to grow as more people support it and more ideas are added to its structure.*

#### **2.1.1.1 Development Background**

The SGA core architecture is a space-function oriented architecture which stresses the operational needs, the applications required to satisfy those needs, the applications' requirements for services to enable them to operate, and the allocation of applications and services to hardware and software. It treats hardware and software as secondary to the operational and services aspect of a space avionics system.

The identification of functions in the SGA core architecture provides an appearance that the SGA core architecture is a software architecture; however, it is intended to represent higher level features comprising both hardware and software. Since all future space data systems will be so heavily dependent on software, this consideration was a primary driver to insure effective software requirements, and especially effective software-to-software and software-to-hardware interface definition, as described later in Section 2.3.2.

#### **2.1.1.2 Architectural Principles Required**

Some of the key principles required of an architecture to be effective in future space systems are:

- abstraction
- information hiding
- inheritance
- modularity
- robustness
- extensibility

These principles were used in the development of the SGOAA, and their definitions are contained in Appendix A.



### **2.1.1.3 Architecture Assumptions and Ground Rules**

Development of this architecture was based on use of computer aided systems engineering tools, as described in [WRA91]. Architecture analysis needs automated tool support. The tools need to provide basic structured analysis capabilities, centered on an integrated data repository. A goal in picking automated tools should be the quality of their support in enabling this methodology to be implemented. A selection factor should be the accessibility of their data repository to other tools so that data from one tool can be extracted and passed to another tool; preferably by using open standards for the data repository structure. No weight should be given to a tool claiming to be capable of performing all phases of automated development, since such a claim is far beyond the state-of-the-art in present tools, and may not be desirable anyway. It seems likely that the tools are secondary to the quality of the analysts in performing architecture analysis; if so, then the tools should be selected to enhance the abilities of individual analysts.

The state-of-the-art, analytical techniques to be used require training to gain understanding of static structured analysis, interface analysis, information modeling, object oriented analysis, control state analysis, timeline analysis, performance analysis, dynamic system modeling, and others used in architectural analysis.

Another key finding is that architectural definition be part of a concurrent engineering approach, requiring the integrated efforts of engineers with experience in several different disciplines, from requirements to design, from hardware to software, and from operation to supportability. Development of an effective architectural model for any specific application or system needs iteration between the concerns of each specialty to insure that the resulting model is responsive to all discipline concerns. Effective techniques to enable multiple disciplines to efficiently interact need to be developed.

Development interface capabilities are needed. Much of the development of requirements for complex space systems is taking place at geographically disbursed contractor sites; these distributed requirements need to be capable of being coordinated on a continuous basis as they are developed. A capability is needed to acquire working level requirements (in process) from these sites, test them against each other and a larger model of the system, and to feed back weaknesses and strengths to the developers of individual requirements sets. This is necessary to avoid waiting too long before erroneous, deficient, weak or conflicting requirements are uncovered; the later the correction of requirements, the greater the cost.

This implies some data repository structure or interface standards are needed to insure that requirements data can be exchanged. It also implies that more frequent technical interchanges would benefit the development of subsystem requirements and architectural elements (or sub-architecture).

An architecture developed for this standard must be adaptable to different platforms and missions, so that architectural interfaces are guaranteed to be interoperable based on the overall architecture from which they are derived. For example, the same architecture should provide the elements needed for a space vehicle operations command function and for the complementary ground processing in the ground mission control center supporting that vehicle.

The architecture development was used as the vehicle for determining what practices actually worked which should be included in this methodology. Assumptions about the architecture were necessary to permit continued development of concepts and entities, and were selected to place as little restriction on the underlying methodology as possible. However, in case they may have constrained the methodology, they are identified below. This section summarizes the architecture assumptions in three categories: those assumptions related to the operation of a space platform, those related to the processing to be performed, and those related to how the structure of the architecture was to be assembled.

#### **2.1.1.3.1 Operations**

- Human control requirements can vary. Direct links from the human entry systems to the sensor and effector firmware/hardware or any intermediate point on the processing chain may be needed for emergency and manual backup purposes. The range of control must accommodate any level of capability from manual to fully automated (e.g., through artificial intelligence aids similar to the Lockheed Pilot's Associate being developed for the U.S. Air Force).
- Operations control requirements must span the range from on-board controls to mission control center to the "back room" control support. The SGOAA must explicitly define and incorporate unique elements for either specific ground support or space vehicle architectural needs. Partitioning between these facility control requirements should be done when applying the requirements to a specific platform

or mission, or should be delayed until a design implementation is being prepared to maximize developer flexibility.

#### **2.1.1.3.2 Process**

- The architecture must enable objective definition and interoperable processes for each entity selected for inclusion in a specific instantiation of the architecture for a specific platform.
- All entities have processes which can be applied to multiple vehicles with control parameters used to adjust between the same type process used in different classes of vehicles.
- Sensors and effectors are assumed to be under hardware control only or to have firmware embedded in them for low level hardware control. Firmware processing is treated (for requirements and design purposes) as an integral part of the hardware. Sensors firmware processing may also enable or disable hardware, monitor power drain, monitor for abnormal conditions, implement built-in-test (BIT) of hardware and store results (these may alternatively be performed in the intermediate processors as described below).
- The architecture must handle alternative forms of processing and alternative allocations of these processes to different elements of the overall system for each mission design. Low level processing is assumed to be embedded in sensor and effector heads; such processing may be hardware only or have embedded firmware such that the processing will be relatively "dumb" but with sufficient capability to gather data, format it for transmission, and route it to appropriate controllers. Intermediate level processing includes processing such as sensor signal processing, effector response actuator processing, post sensor processing (e.g., track processing), multiplex data processing etc.; such processing is treated as a high level control structures (i.e., Control Application Programs) requiring some decision making capability to implement one of a number of alternative hardware control parameter sets in an intelligent system. High level processing includes two types of processing: one which provides a capability for the crew to control the vehicle or facility, and one for internal systems control of all activities. Processing such as needed for systems control, vehicle control, integrated logistics control, crew management, etc. are treated as high level command structures (i.e., Command Application Programs), which require interaction with humans and some capability to present alternatives

to humans, and to interpret ambiguous responses from humans. Command application programs provide both types of high level processing. Figure 2-4 depicts the processing architecture assumed which the Space Generic Architecture must handle, and which the methodology for development must be capable of analyzing.

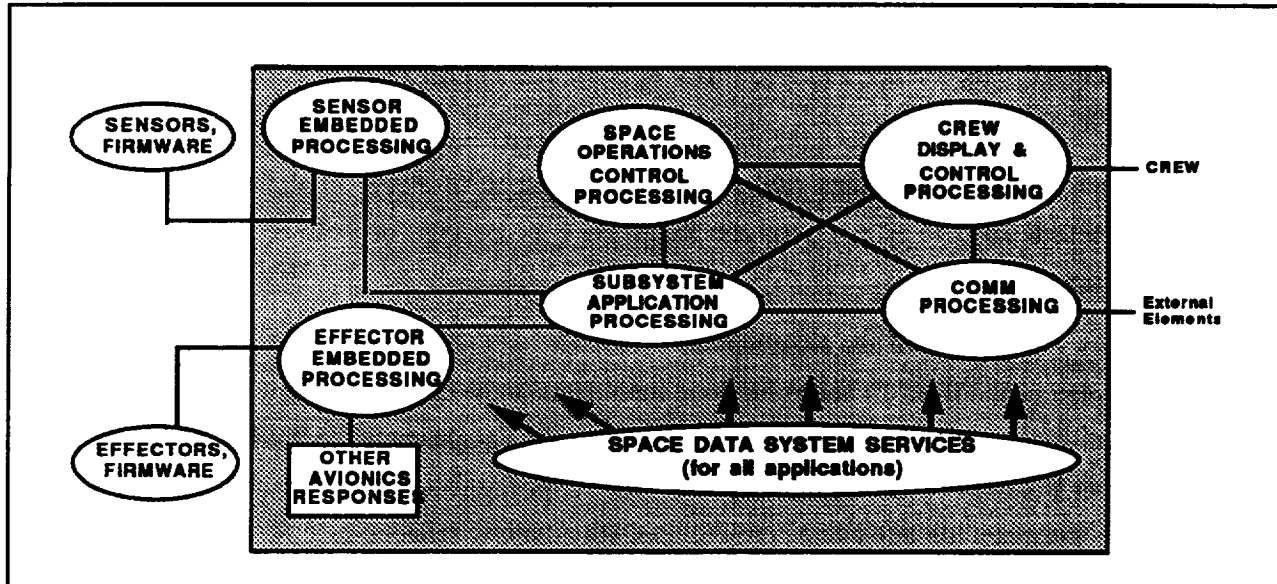


Figure 2-4. Potential Process Partitioning included by the Space Generic Architecture

#### 2.1.1.3.3 Structure

- One of the purposes is to enable the creation of an open, generic, standard architecture which can be tailored and reused for multiple missions. The methodology will then provide guidelines for doing the tailoring to create mission specific instantiations of the architecture. The reuse of the architecture and its components will become the standard way of developing new space data systems.
- The basic architectural guideline for differentiating processing levels is based on the philosophy of "Centralized Command and Decentralized Execution"
- The architecture must be a "shopping list" of all possible processes applicable to any space vehicle or other-planet base.
- Some entity processes only apply to a specific class of vehicle. Such special entity applications should be built into the naming conventions if feasible to more clearly convey the dependency of the entity application to the specific platform. The

convey the dependency of the entity application to the specific platform. The definition of entity names must use unique names for each entity for clarity and for tool searching of dependencies.

- The software principles of abstraction, information hiding and modularity are applicable to systems development and will provide the same benefits to requirements analysis and system design as they do to software analysis and design. Use of such principles will improve the maintainability and reusability of the architecture developed and used as the example for this methodology. Improvements in maintainability and reusability will not be allowed to reduce the requirements for performance which may be necessary; proof through architectural simulations must be provided that performance of an architectural instantiation is acceptable. Hard real-time constraints on system performance will exist and must be met.
- A hardware architecture was assumed consisting of a core network, multiple general avionics processors (GAP) elements not necessarily of the same type, multiple buses, multiple multiplex data processing [GAP(M)] elements, embedded sensor processing [EP(s)] and embedded effector processing [EP(e)]. This is represented in Figure 2-5 . The interface plugs shown represent the unique hardware interfaces which must be defined by standards and handled in processing.

### **2.1.2 REFERENCE MODEL DEFINITIONS**

Definitions of the terminology used in this architecture standard are contained in Appendix A. The terminology is based on industry standards definitions wherever feasible.

Determination of the scope of architecture, avionics, systems, services and applications depends to some extent on the definitions for these items since definitions can focus attention or exclude attention. The reference model must describe information interfaces using services available to an Application from the host Application Platform. It must describe processing interfaces using services available internally to Applications Platforms, meeting the requirements of Applications Software entities. Processing interfaces provide the services commonly available to many applications, but not designed for just one application.

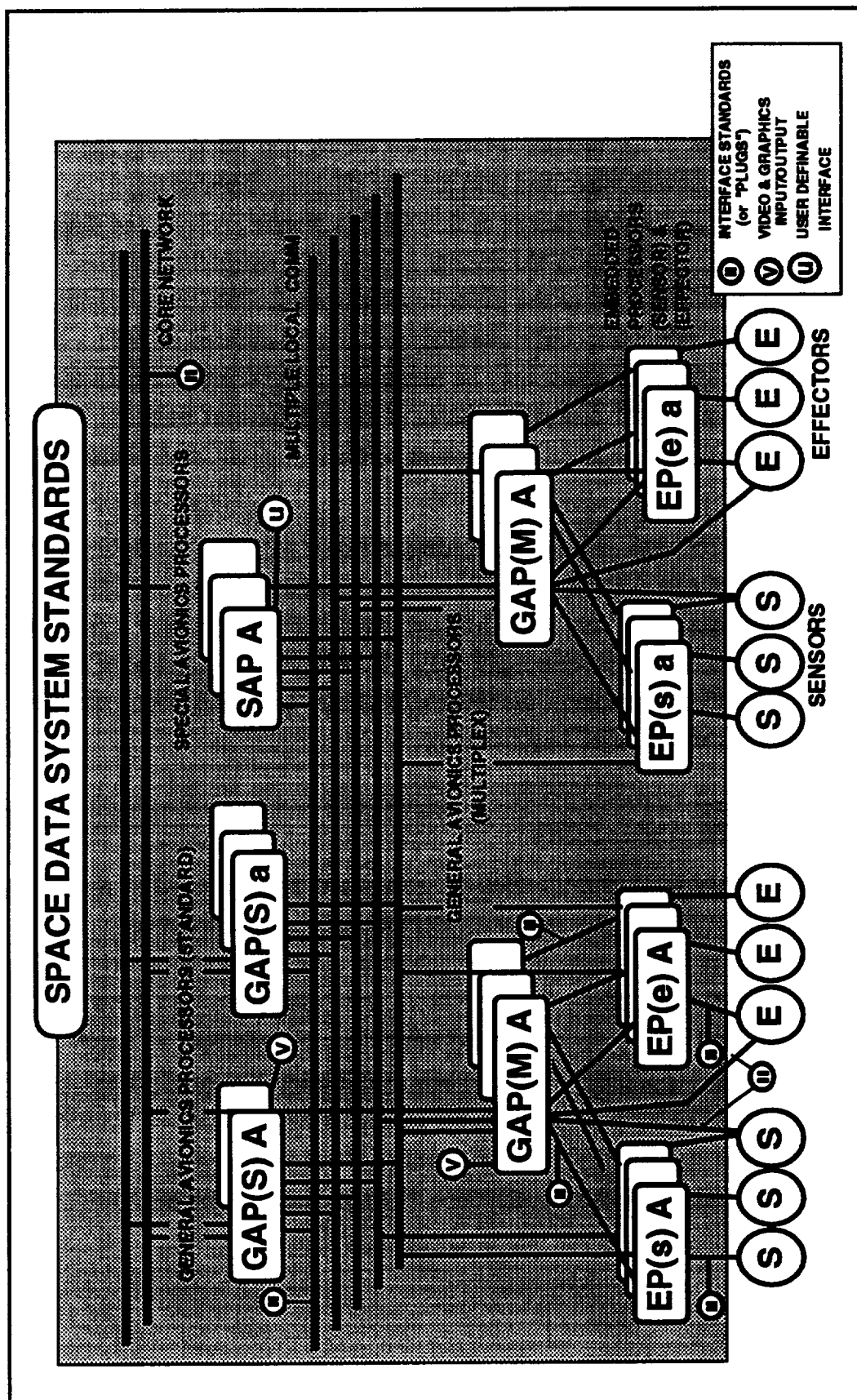


Figure 2-5. Hardware Architecture Assumed for the Space Generic Avionics

### 2.1.3 RELATED ARCHITECTURAL STANDARDS AND REFERENCE MODELS FOR AVIONICS

#### 2.1.3.1 POSIX Open System Environment

The POSIX OSE Reference Model, defined in [POSIX91], is the top level standard within which the SGOAA must fit, as previously shown in Figure 2-1. The OSE Reference Model enables application software portability at the source code level and application software and system service interoperability between heterogeneous systems. This environment will establish a set of specifications, standards and procedures common to all missions which must operate concurrently, with inherent upgradeability. Definition of entities and interfaces based on the OSE model can facilitate requirements definition for designs which have the open and generic characteristics needed. This model is not an implementation architecture; it is used to identify subsystems (entities in the model), interfaces between subsystems, and services at the interfaces. Figure 2-6 depicts the OSE model.

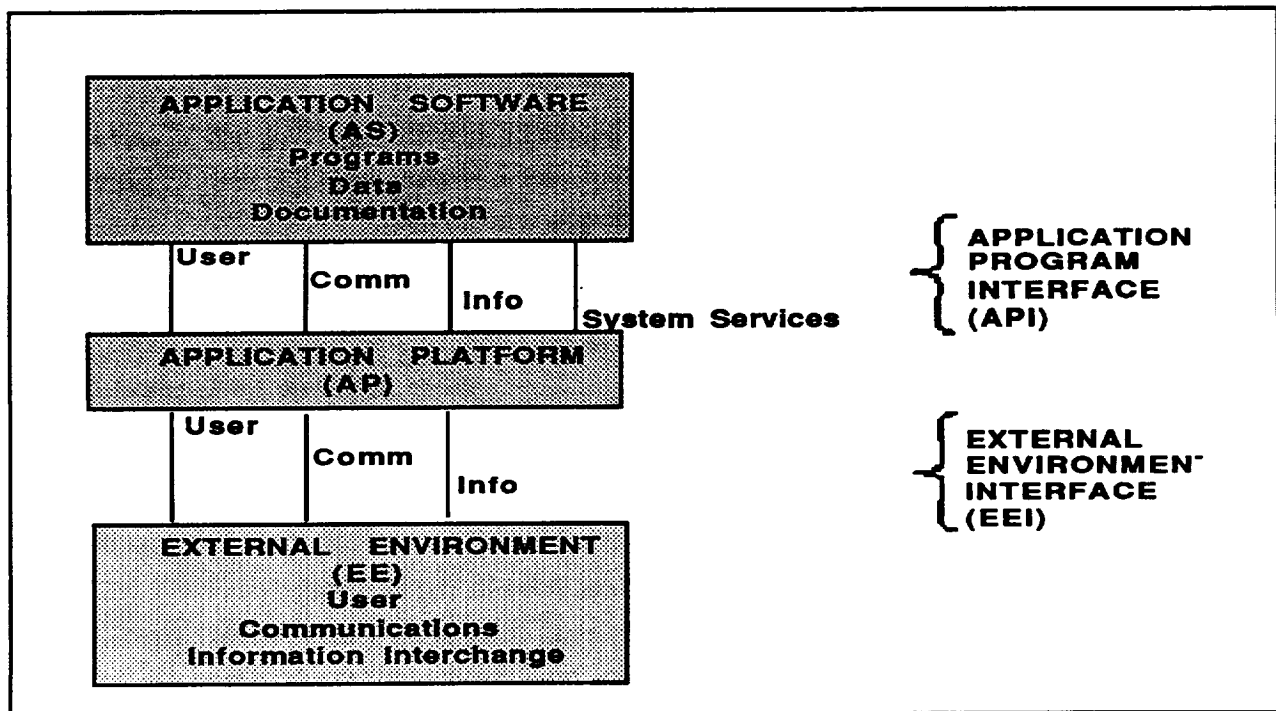


Figure 2-6. Open System Environment Model of Applications and Interfaces

There are three types of entities used in the OSE Model: Application Software, Application Platform and External Environment. Application Software (AS) is the set of processes, data and associated performance parameters and documentation in electronic form related to operation of a data processing system. An Application Platform (AP) is the set of services and resources needed to run the applications. The External Environment (EE) is those elements outside the boundaries of the entity of interest which need to exchange information with the entity of interest. The external environment includes permanent data stores, electronic communications entities and human entities.

Applications Software interfaces through the Applications Program Interface (API) to the Application Platforms. The API interfaces are:

- User - An interface intended to provide access from the Applications Software with the user by defining the services available to the applications software for exchanging information with the user.
- Communications - An interface defining services available to the Applications Software to exchange state and information with Applications Platforms or other Applications Software.
- Information - An interface providing non-communications language bindings and services to exchange state and information to be provided with the Applications Platforms or with other Applications Software.
- System Services - An interface defining language bindings and services for available internally to the Applications Platforms and not used by the Applications Software for portability or interoperability with other Applications Software or Applications Platforms.

Application Platforms interface through the External Environment Interface (EEI) to other Application Platforms. The three types of interfaces used in the EEI are:

- User - An interface for physical access between the machine and human, providing the "look and feel" of the means of human interaction with the Application Platforms.
- Communications - An interface providing language bindings for service for connectivity and protocols for state and data interchange.
- Information - An interface providing language bindings for service using physical and logical file structures, characterized by floppy media.



The OSE model, shown in Figure 2-6, using AS, APIs, APs, EEIs and the EE can involve multiple subsystems. In the SGOAA, each subsystem application (e.g., GNC Control, C&T Control and SOCS) is the Application Software. The central architecture consisting of processing hardware and system services are the APs. The subsystem application to system services interface is the API, which is implemented for communications through the SDSS communications network services at the OSI layer 7. The AP (i.e., avionics) to EE (i.e., the users, hardware sensors, effectors and communications devices) interface is the EEI, which is implemented for communications through the SDSS communications network services at the OSI layer 1.

Section 2.4 discusses the relationships of the SGOAA Interface Model Classes to the POSIX Model interfaces.

#### **2.1.3.2 OSI Model**

The OSI Reference Model [ISO7498] is a Network Services Model. Network Services is only one resource of many competing resource processes provided by both the POSIX and SGOAA Models. Applications gain access to POSIX Network Services via the POSIX API Communications Services Interface and to SGOAA Network Services via the SGOAA Class 5 Interface (Applications Software-to-System Services Software). SGOAA Network Services are discussed in more detail in paragraph 3.4.1.3. In the OSI model, applications gain access to Network Services via an applications-to-services interface. Interfaces provided by Network Services must be open network interfaces, protocol independent and provide for network protocol interoperability. The POSIX OSE reference model focuses on the requirements of application portability and system interoperability at the source code level by addressing these objectives at the Applications Program Interface (API) and at the EEI. Internal Application Platform Interfaces are not addressed.

The OSI Model defines the Communications (Network) Services API and EEI interfaces of the POSIX OSE reference model and the SGOAA Interface Model as shown in Figure 2-1. The OSI Model expands upon the POSIX Communications Services Interface and SGOAA Network Services by defining in great detail how Network Services Standards should work and fit together. SGOAA extends the POSIX Model beyond the basic POSIX objectives by defining six SGOAA interface classes, addressing application platform internal interfaces and recommending additional data systems software specifically applicable to space based

systems. The compatibility of the OSI model and the SGOAA Interface Model involves the interface class relationships shown below:

- OSI Layer 1: This layer is the actual connection to the transmission medium, handling the transmission and reception of raw bits across the medium. This is a SGOAA Class 1 Hardware-to-Hardware Direct interface.
- OSI Layer 2: The interface between OSI hardware and software layers. This is a SGOAA Class 2 Hardware-to-System Software Direct interface.
- OSI Layer 3: A software layer that accepts packets (or frames) of data from the Transport Layer (software) and routes them to their destination over all necessary links and intermediate systems as necessary. This is a SGOAA Class 4 System Software-to-System Software Logical interface.
- OSI Layer 4: A software layer that provides reliable data flow between a sender and a receiver while relieving these entities of the need for detailed knowledge of the actual transport mechanism. This is a SGOAA Class 4 System Software-to-System Software Logical interface.
- OSI Layer 5: A software layer that establishes communications paths between systems and terminates them upon completion of transmissions. This is a SGOAA Class 4 System Software-to-System Software Logical Interface.
- OSI Layer 6: A software layer that performs a translation function to convert messages from a native format to an international standard format for transmission, and from the international format to the native format upon reception. The international format is a transfer syntax, a set of rules for the representation of data while in transit between two presentation entities. This translation is performed by Network Services on network data only and not to application data. This is a SGOAA Class 4 System Software-to-System Software Logical interface.
- OSI Layer 7: Provides the interface between application programs and the network. This is a SGOAA Class 5 System Software-to-Application Software Direct interface.

The OSI model does not address SGOAA Class 3 System Software-to-Software (Local) Direct nor does it address SGOAA Class 6 Application Software to Application Software Logical interfaces.

Much standardization effort has gone into all aspects of networking, especially those aspects that are available at the EEI. Effective networking standards at the external interface are fundamental to providing system interoperability.



## **2.2 SPACE GENERIC OPEN AVIONICS ARCHITECTURE REQUIREMENTS**

These requirements are based in part on the generic open architecture requirements defined in reference [BOE91]. These requirements provide a baseline for tailoring to a specific mission or vehicle. Specific performance parameters from [BOE91] presented in Appendix B are design requirements and as such are outside the scope of the SGOAA. Additional requirements were added from other sources and experience to make the SGOAA as generic and broad as possible.

The SGOAA shall be used to determine the interface points and requirements for the control of, and information exchange between, onboard subsystems, support to the crew, and effective interfaces between onboard and offboard systems. In accordance with system requirements, a SGOAA compliant architecture shall meet open standards criteria.

### **2.2.1 REQUIREMENTS OVERVIEW**

The SGOAA shall provide for the control and information processing of onboard subsystems, support to the crew and effective interfaces between onboard and offboard systems. A SGOAA compliant system architecture shall provide data acquisition, data storage, data processing and data communication functions that interconnect architectural elements as shown in the functional interface diagram (see Figure 2-7). The SGOAA shall also provide the capabilities to implement, where required, data base access, electronic mail, planning, training, simulation and monitoring of payload interfaces. Architectures developed in accordance with the SGOAA shall meet the requirements of [WRA93] for developing new architectural elements and for using existing applications and mission elements.

#### **2.2.1.1 Open Systems Requirements**

An architecture developed in accordance with this standard shall satisfy the open systems architecture definition incorporated in this standard. An architecture shall meet open standards to insure access to the architecture for any vendor or agency desiring to propose new uses and applications, and to facilitate competition to contain cost growth. The open architecture so developed shall be capable of being readily expanded in functionality and performance without redesign or significant modification to the existing system. An architecture satisfying this standard shall provide information hiding, abstraction, inheritance, modularity, robustness and extensibility.

Control subsystems may be decomposed into lower level subsystems. A control subsystem usually implements a unique avionics capability.

#### **2.2.1.2 Lower Level Standard Selection**

Lower level standards developed by accredited standards development organizations (which use an open forum) shall be preferred in selection over those standards developed by bodies using a closed forum. Lower level standards shall be selected by the process of developing a standardized profile. Architecture specifications for which there is no draft or approved standard shall not be selected. One of the driving requirements for selection shall be selection of a standard that provides the full range of services required to satisfy the system applications. Other factors to consider in standards selection shall be degree of openness in development, stage of completion, stability, compliance with national and international standards, degree of satisfying a SGOAA service need, consistency with the SGOAA and availability for implementation without restrictions.

Preference shall be given to existing mature standards, followed by emerging standards, and only if necessary, followed by new standards. The order of selection within these preferences is as follows:

- Approved standards developed by (a) accredited international bodies, (b) accredited regional bodies and (c) accredited national bodies.
- Draft standards developed by (a) accredited international bodies, (b) accredited regional bodies and (c) accredited national bodies.
- Recognized de facto standards and specifications developed by nonaccredited bodies using an open forum.
- Approved standards and specifications developed by nonaccredited international standards bodies using a closed forum.
- Approved standards and specifications developed by nonaccredited national standards bodies using a closed forum.

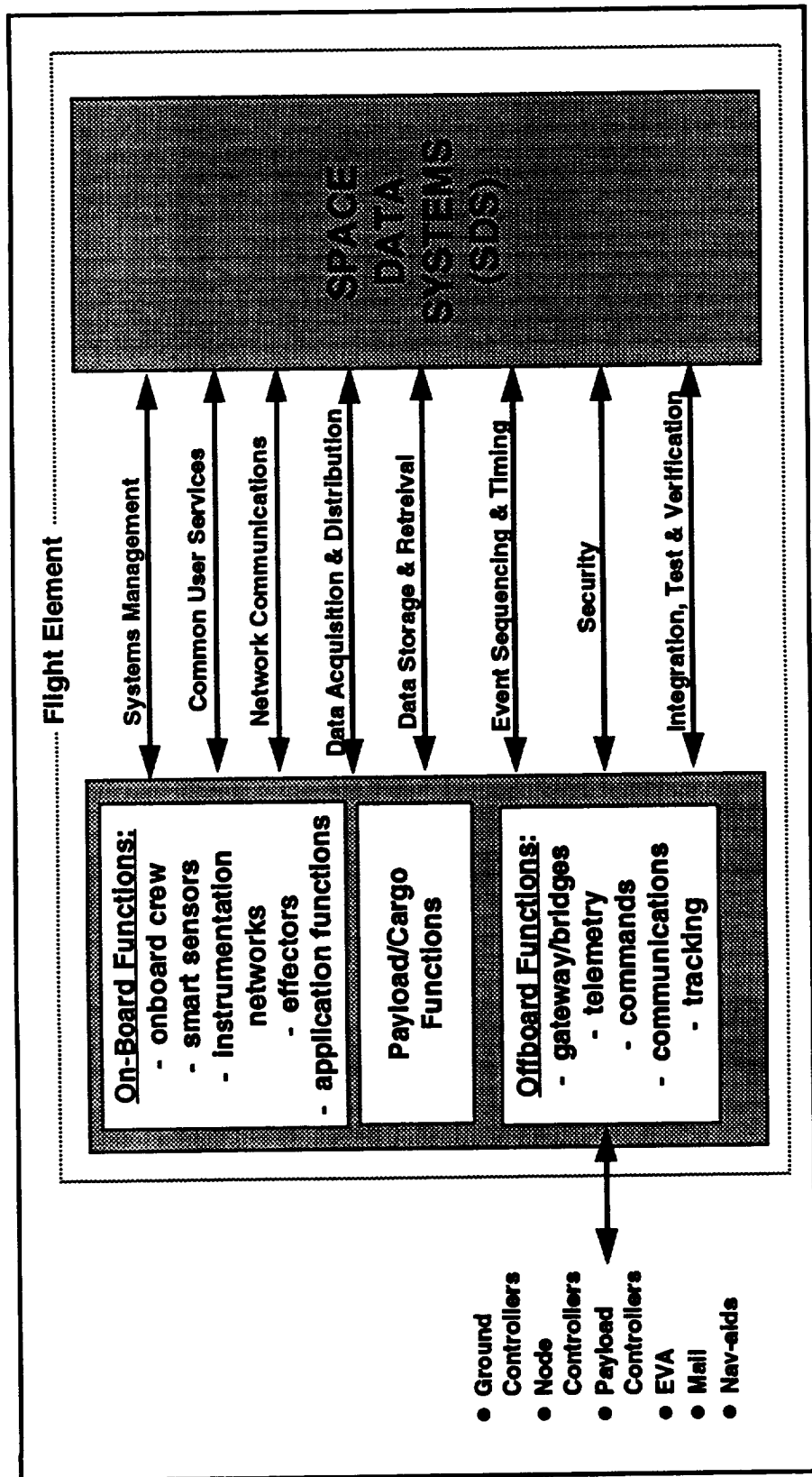


Figure 2-7. SGOAA Functional Interfaces

## **2.2.2 ARCHITECTURE FEATURES**

Requirements for architectures compliant with the SGOAA consist of general guidelines for developing new architectural elements and for using existing elements in tailoring architectures to specific applications and missions. Developing a capability to apply generic or other standard architectures to new missions and vehicles is a key focus of SGOAA requirements definition. Requirements for SGOAA compliant architectures also address detailed, specific technical features which must be achieved by acceptable architectures. This section describes the features which a SGOAA compliant architecture must provide:

Privacy and proprietary data will be handled by the SGOAA, with provisions and interfaces for handling national security data requiring NSA-type protection requirements to be optional.

### **2.2.2.1 Requirements Architecture**

An architecture prepared in accordance with this standard shall be an architecture that can be tailored for design implementation based on actual system requirements.

### **2.2.2.2 Critical Interfaces**

An architecture prepared in accordance with this standard shall provide flight, mission and safety critical functions and interfaces.

### **2.2.2.3 Service Interfaces**

An architecture prepared in accordance with this standard shall provide non-critical support functions and interfaces, such as data base access, electronic mail, planning, training, simulation and monitoring of payload interfaces.

### **2.2.2.4 Resource Control**

An architecture prepared in accordance with this standard shall provide for control of system resources used for control and information processing in onboard systems by use of system services software as requested by application software through a standard interface.



#### **2.2.2.5 Commonality**

The architecture shall be comprised of common hardware and software components to the maximum extent possible. The SGOAA shall require the use of standard interfaces. Non-common components or non-standard interfaces shall require a waiver from the procuring agency.

#### **2.2.2.6 Interface Standardization**

An architecture prepared in accordance with this standard shall provide standard interfaces and allow user definable interfaces where no standards exist or are not applicable. Interfaces between hardware and other hardware entities shall be based on standards. Interfaces between hardware and software shall be based on standards. Interfaces between system services software and applications software shall be based on standards. Interfaces prohibited in an architecture compliant with this standard shall include: (1) direct, non-service task to task communications, and (2) applications to applications direct information exchanges, which bypass use of system services. Special user definable interfaces may be defined within the standards.

SGOAA allows external interface(s) for:

- gateway to/from non-SGOAA component
- bridge to/from non-SGOAA component
- data to/from Communications and Tracking
- onboard crew user interfaces
- transducer interfaces (sensors and effectors).

#### **2.2.2.7 Crew Override**

For crewed vehicles, an architecture prepared in accordance with this standard shall enable crew intervention, through multiple techniques, to safely override or inhibit automatic flight, mission or safety critical functions. For uncrewed vehicles, the architecture shall enable ground control station intervention to safely override or inhibit flight, mission or safety critical functions.

### **2.2.2.8 Dependability Management**

An architecture prepared in accordance with this standard shall manage dependability.

An architecture compliant with this standard shall provide at least health and status monitoring and warning capability to monitor critical functions in onboard systems, subsystems, components and crew. System service software BIT shall be incorporated into software control modules. Hardware built-in-test equipment (BITE) shall be incorporated into hardware modules. The interface between hardware BITE and health and status applications software shall be through standard software services.

An architecture compliant with this standard shall provide at least operating modes for: (1) mission ready, (2) operationally ready, (3) degraded, and (4) red-tagged.

### **2.2.2.9 Data System Services**

An architecture prepared in accordance with this standard shall include requirements for data system services. This shall consist of at least requirements for standard data services management, network services management, data base management, data system management, and an operating system.

The standard data services management shall include at least requirements for standard services data acquisition, standard services data distribution and reports generation. The network services management shall include at least requirements for network services, network management, remote operation, network directory service, and network association control. The data base management shall include at least requirements for file services, distributed file transfer services, file transfer access and management, and node directory. The data system management shall include at least requirements for configuration management, timing service control, initialization startup and reconfiguration, and health status and fault detection and recovery. The operating system shall include at least requirements for an OS kernel, an Ada run time environment (RTE) and OS/RTE extensions.

An architecture prepared in accordance with this standard shall support onboard fault recovery.

#### **2.2.2.10 Growth and Spare Capacity**

An architecture prepared in accordance with this standard shall accommodate growth and spare capacity in data storage, processing throughput, network throughput, input/output and additional sensors/actuators as required by system documentation.

#### **2.2.2.11 Modularity**

An architecture prepared in accordance with this standard shall be modular.

#### **2.2.2.12 Service Transparency**

An architecture prepared in accordance with this standard shall be implemented with sufficient transparency that the user will have visibility into the operation of services, but not necessarily the implementation of services.

#### **2.2.2.13 Technology Transparency**

An architecture prepared in accordance with this standard shall be implemented with sufficient transparency that technologies applied to design can be upgraded without revising the architecture and without negative impact on the user.

#### **2.2.2.14 Interoperability**

An architecture prepared in accordance with this standard shall support interoperability by providing standard interfaces between multiple programs.

#### **2.2.2.15 Goals**

The following goals are desirable characteristics that a system architecture should possess. They are not incorporated into the SGOAA Standard as they are not considered to be mandatory requirements.

- (1) An architecture should be sufficiently general and portable to be adapted and applied to many missions, platforms and vehicles, meeting many operational requirements, to enable designs for new missions to be prepared with less cost growth and with more reasonable development schedules (by reusing existing architecture, hardware and software components).

- (2) An architecture should facilitate design structures which can be verified (by establishing consistency and completeness across multiple platforms)
- (3) An architecture should provide a basis for validating that systems and procedures meet future space mission needs
- (4) An architecture should ensure future space avionics systems can be upgraded and maintained through use of modular interfaces and reuse of hardware and software
- (5) An architecture should have integrated the end-user avionics and support data processing and controls.
- (6) The design and development approach should not unduly constrain the architecture and its application.
- (7) The architecture shall provide for optional capabilities selectable by the users during design, such as event timing to sequence events as determined by application requirements.
- (8) The architecture shall facilitate migration of functions from offboard (ground or node) to onboard components.
- (9) The architecture shall provide for the separation of functions and resources for the levels of functional criticality shown in Figure 2-8 and defined in Table 2-1. Note that the notation and definitions in Table 2-1 are used throughout this section to segregate functional requirements. The notations are as follows:
  - Flight Critical, Level 1 = FC1
  - Safety Critical, Level 1 = SC1
  - Mission Critical, Level 2 = MC2
  - Utility, Level 3 = NC3

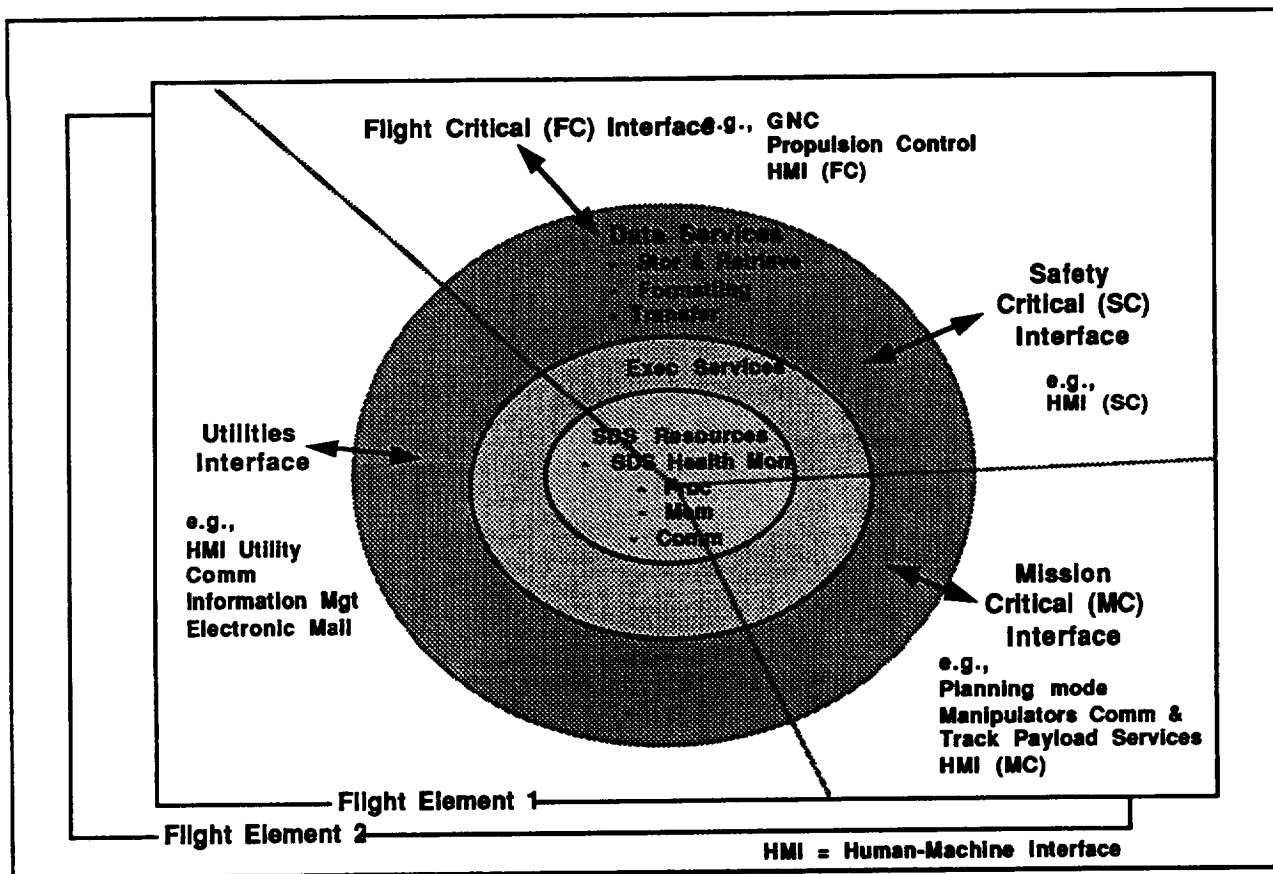


Figure 2-8. SGOAA Functional Requirements

Table 2-1. Critical Condition Categories

Level	Category	Critical Condition
1 (high)	Flight Critical (FC)	Loss of vehicle control results in loss of vehicle and crew.
1	Safety Critical (SC)	Increase in hazard results in loss of vehicle, crew or both
2	Mission Critical (MC)	Incomplete mission results in mission abort or loss of payload
3	Utility (NC)	No control loss or unsafe condition

## **2.2.3 EXTERNAL INTERFACES**

Section 2.2.3.1 contains the specification of requirements for the interfaces of a system architecture with its external environment as derived from the SGOAA. The standard interfaces shall include the direct/physical and indirect/logical interfaces to meet functional system requirements.

Section 2.2.3.2 contains objectives for providing system capabilities that the development team considered in development of the SGOAA.

### **2.2.3.1 External Interface Requirements**

Portability, interoperability and standards usage are mandatory requirements included in the SGOAA Standard.

#### **2.2.3.1.1 Portability**

Portability is accomplished by a standard interface between executive software and application software. Direct physical interface application software to application software communication is not allowed.

#### **2.2.3.1.2 Interoperability**

Interoperability is provided by standard interfaces between software applications and between application platforms. The standard interfaces include the physical, electrical and optical interconnection and both the logical and physical functional interfaces.

#### **2.2.3.1.3 Standards**

The SGOAA will accommodate existing, emerging and new information technical standards. The recommended selection of standards shall be consistent with the applicable NASA standards such as NASA-STD 3000, Volume IV and shall also comply with the [POSIX91] recommendations. A partial list of related standards is shown in Table 2-2. Selection will draw from many different sources. The selection of lower level standards is discussed in paragraph 2.2.1.2.

Table 2-2. Partial List Of Related Standards

<b>STANDARD NO.</b>	<b>TITLE</b>	<b>SOURCE</b>
9945-1	Process Management	International Organization for Standardization and International Electrotechnical Commission (ISO/IEC)
MIL-STD-1553	MIL-STD-1553 Multiplex Application Handbook	United States Air Force
X3.168	SQL Standard Database Language	American National Standards Institute (ANSI)
P1003.1	POSIX System Interfaces	Institute of Electrical and Electronic Engineers (IEEE)

#### **2.2.3.2 SGOAA Development Interface Definition Objectives**

In development of the SGOAA, it was a requirement to provide the SGOAA with the capability to support the interfaces defined in this paragraph when applying the SGOAA to a specific system design.

##### **2.2.3.2.1 Human Interfaces**

For occupied manned elements, the SGOAA shall provide interfaces for the following avionics to human interfaces and resource capabilities.

##### **(1) Onboard Human interfaces**

Onboard crew interface to the avionics for assembly/checkout, pre-mission, mission and post mission phases.

Real-time (FC1, SC1,MC2) processing and interface for controls and displays.

Processing and interfacing to controls and displays of critical non-real-time functions (SC1,MC2).

Processing and interfacing to manned systems desktop functions and electronic mail.

Resources and interfaces for application development and modifications.

Manned monitoring of subsystem status.

Resources and interfaces for training and simulations

## **(2) Offboard Human Interfaces**

Offboard human interface to the avionics for assembly/checkout, pre-mission, mission and post mission phases.

An external control center command and control interface.

An external control center payload/cargo data interface.

### **2.2.3.2.2 Application Interfaces**

The SGOAA requires that the only interfaces an application has is through a standard interface and the subsystems the application supports. No application-to-application physical interface is allowed. The standard interface shall support messages, datagrams (non-acknowledged messages), files and records. Critical real-time software must be deterministic for both operations and for verification. Application data traffic for both operations and for verification shall meet the priority requirements of Data Handling Capability. The SGOAA shall provide the following support to applications:

Software support:

- critical, real-time (FC1,SC1, MC2)
- critical, non-real-time (SC1,MC2)
- non-critical (NC3)

Hardware interface:

- critical, real-time (FC1,SC1, MC2)
- critical, non-real-time (SC1,MC2)
- non-critical (NC3)

Operations access control:

- critical, real-time (FC1,SC1, MC2)
- critical, non-real-time (SC1,MC2)
- non-critical (NC3)

System health monitoring data for all onboard functions.



#### **2.2.3.2.3 Sensor Interfaces**

The SGOAA shall support interfaces for the implementation of signal conditioning for either analog or digital sensor signals. The SGOAA shall support either direct connection to the sensor interface or through an intermediate network. The standard sensor interfaces shall be of a data bus compatible, data bus non-compatible or high data rate type and shall accommodate both real-time and non-real time data.

#### **2.2.3.2.4 Effector Interfaces**

The SGOAA shall support interfaces for the implementation of signal conditioning for analog and/or digital effectors. The SGOAA shall support either direct connection to the effector interface or through an intermediate network.

#### **2.2.3.2.5 Payload/Cargo Interfaces**

The SGOAA shall support a standard interface to payload(s). The standard interface shall include:

- A standard interface for the command and control of payload(s) and cargo.
- Standard data communications from payloads and cargo to offboard interfaces.
- Payload event timing for employment of each onboard payload.

### **2.2.4 SGOAA DEVELOPMENT FUNCTIONAL REQUIREMENTS**

The SGOAA shall be designed to support systems based upon a modular building block approach.

Instantiations of the SGOAA shall share common functional building blocks in order to minimize the number of different types of functional building blocks. The SGOAA shall provide standardized hardware interfaces for common functions. The SGOAA shall provide standard Operating System interfaces for application software.

### 2.2.4.1 Process and Data Requirements

#### 2.2.4.1.1 Message Transfer

The SGOAA shall provide the capability to implement multiple service grades for the levels of criticality shown in Table 2-3 and for onboard users to request each grade of service. RT is real-time service and NRT is non-real-time service.

Table 2-3. Message Transfer Service Grade

FUNCTION	GRADE		
	I	II	III
FC1(RT)	Required	not allowed	not allowed
MC2(RT) MC2(NRT)	Desired Allowed	Allowed Desired	not allowed Allowed
SC1(RT) SC1(NRT)	Required Required	not allowed not allowed	not allowed not allowed
NC3(NRT)	Allowed	Allowed	Desired

Three grades of data delivery service are to be supported, as defined in Table 2-4, among onboard users and between users and Communications and Tracking (C&T).

Table 2-4. Message Transfer Service Grade Characteristics

CHARACTERISTIC	GRADE		
	I	II	III
Error Detection	yes	yes	no
Out of Sequence Check	yes	no	no
Erroneous Duplicate Check	yes	no	no
Completeness Check	yes	no	no

#### 2.2.4.1.2 Data Storage and Retrieval

The SGOAA shall provide for the support of standard data transfers within the SGOAA. For transfers outside of SGOAA boundary , one end of transfer will be from a gateway or C&T and other end will be SGOAA.

The SGOAA shall provide for support of a Mass Data Storage Capability.

In support of file/record management, the SGOAA shall provide for support of Global Standard File Naming and Directories.

In support of data base management , the SGOAA shall provide for local and remote access to onboard SGOAA databases that is consistent with security and data privacy requirements.

The SGOAA shall provide for the support of an Electronic Mail Service standard.

#### **2.2.4.1.3 Crew Common User Data Services**

The SGOAA shall provide for the implementation of resources and interfaces to support the following manned flight crew common user data services:

- Monitor and control access interface to avionics for manual override or inhibit of all functions.
- Monitor and control access [MC2(RT), MC2(NRT), SC1(RT) and SC1(NRT)] interface to payloads for manual override or inhibit of all functions.
- Caution and Warning Displays [SC1(RT) and SC1(NRT)] interface for audible/auditory/visual alarms.
- Utilities [NC3(NRT)] interfaces to support on-line help, word processing, mail, calculator, spreadsheet, display processing.
- Unique avionics support to training [NC3(NRT)]
- Unique avionics support to simulation, real-time [SC3(RT)] and non-real-time [SC3(NRT)] resources and interfaces for in-situ skill training.

#### **2.2.4.1.4 Data Systems Management**

The SGOAA shall provide for support of fully automatic operation for each mission phase with human intervention via authenticated crew input.

For health and status reporting, the SGOAA shall provide the capability to implement data gathering from all onboard functions.

The SGOAA shall provide for the capability to implement safe power up and graceful power down of avionics during initialization and shutdown.

The SGOAA shall provide the capability to control and manage avionics resource usage based upon operations sequencing management and health/status. Support to implementing control requirements for the following areas shall be provided:

- Avionics System configuration control
- Data System Services configuration control
- Avionics System caution and warning
- Integration and reconfiguration of avionics resources
- Network management
- Inventory and maintenance management

#### **2.2.4.1.5 Application Processing Support**

The SGOAA shall provide for support of application processing and interfunctional application data. Executable memory loads shall be divided between flight/safety critical load, mission critical/payload load, and utility load.

#### **2.2.4.1.6 Event Sequencing and Timing**

The SGOAA shall support provision of an event sequencing and timing service interface.

#### **2.2.4.1.7 Data Acquisition and Distribution**

The SGOAA shall provide support of standard data reception, conversion, formatting, transmission, validation and status for:

- Data bus compatible transducers
- Non-compatible type transducers
- Telemetry data formatting
- Instrumentation data acquisition
- High bandwidth data link (s)
- Time reference generation and distribution

#### **2.2.4.1.8 Data Handling Capability**

The SGOAA shall provide for support of user access to user-provided transducers. The SGOAA shall support distribution of payload data to onboard and offboard users through either umbilicals or C&T. The SGOAA shall provide for support for on-line, rapid access mass storage for payload and onboard use.

The SGOAA shall support transparent reception, transmission, processing, storage and distribution of payload commands and data.

The SGOAA shall support implementation of flow and congestion control. Flow and congestion control is defined as the process for detection and correction of congestion in order to prevent the congestion from propagating from a network into other networks. The SGOAA shall support implementing a a priority system for message handling.

#### **2.2.4.1.9 Test and Verfication Support**

The SGOAA shall support the ensuring of end-to-end functional correctness within the SGOAA boundary. The SGOAA environment shall support the necessary tools, services, diagnostics, built-in test equipment, on module built-in test, test plans, and facilities for evolving flight element avionics from assembly, pre-mission checkout through mission operation.

The SGOAA shall support avionics hardware reintegration (introduction of flight hardware previously removed from operation) without requiring shutdown of the total avionics system.

The SGOAA shall support implementation of the necessary tools for the crew to perform onboard or offboard maintenance.

#### **2.2.4.2 Performance and Quality Engineering Considerations**

The following factors should be considered in any system design. The actual requirements must be based upon mission needs. Development of the SGOAA shall be such that it does not preclude support of these considerations. A SGOAA compliant system shall have as a minimum the throughput and memory capacity margins shown in Table 2-5.

Table 2-5. Capacity Margins For Growth\*

<b>CONDITION</b>	<b>PDR</b>	<b>CDR</b>	<b>ACCEPTANCE</b>
% Worst case memory usage of allocated space within resource processor(s)	50%	65%	85%
% Worst case usage of each target processor	5%	50%	65%
% Worst case channel throughput allocation of each network	50%	65%	85%

\*Source - SSP 30000, "Program Definition and Requirements Document"

##### **2.2.4.2.1 Fault Tolerance**

As a minimum, an SGOAA compliant system shall maintain normal operational state in presence of:

- (1) up to two non-simultaneous faults for flight critical functions FC1(RT) within the interface boundaries of the SGOAA.
- (2) one fault for mission critical functions [MC2(RT), MC2(NRT)] within the interface boundaries of the SGOAA.
- (3) no single avionics failure within the interface boundaries of the SGOAA shall cause a safety critical (SC1) function to execute.

As a minimum, an SGOAA compliant system shall maintain normal operational state during periods of in-space assembly and no single avionics failure shall cause a safety critical (SC1) function to execute.

As a minimum, an SGOAA compliant system shall maintain normal operational state during maintenance actions defined as:

- (1) one fault for flight critical functions [FC1(RT)]A
- (2) one fault for mission critical functions [MC2(RT) and MC2(NRT)].
- (3) no single avionics failure shall cause a safety critical (SC1) function to execute.

Non-critical avionics functions (NC3) shall fail in a safe mode.

#### **2.2.4.2.2 Fault Detection, Isolation and Recovery**

SGOAA compliant systems "Onboard Fault Detection Coverage" shall detect at least 99% of faults during power-up and initialization, 90% of faults during background normal operation, and 95% of faults during directed health monitoring tests. (These values may be adapted to or superseded by specific mission requirements. For example, human-tended systems will require considerable higher FDIR than automated, expandable systems.)

SGOAA compliant systems "Onboard Fault Isolation Coverage" for 100% of the detected faults at the module level shall be at least 98% during power-up and initialization, 98% during normal operation and 98% during directed health monitoring tests .

Onboard Fault Recovery TBD.

#### **2.2.4.2.3 Reliability**

SGOAA compliant systems shall achieve the Mean Time Between Critical Failure (MTBCF) and Mean Time Between Failure (MTBF) as listed in table 2-6. Confidence factor is the proportion of times that the parameter lies within confidence interval. The actual values that are selected to replace the TBDs in the table will be design requirements, not SGOAA requirements, and must be based on mission needs.

Table 2-6. Reliability Requirements

FUNCTION	MISSION RELIABILITY	BASIC RELIABILITY	CONFIDENCE FACTOR
	(MTBCF)	(MTBF)	
FC1	TBD	N/A	99%
MC2(RT)	TBD	N/A	95%
MC2(NRT)	TBD	TBD	95%
SC1(RT)	TBD	N/A	99%
SC1(NRT)	TBD	TBD	99%
NC3	N/A	TBD	95%
overall system	TBD	TBD	95%

#### 2.2.4.2.4 Availability

SGOAA Availability is defined for two entities: A and Ai. SGOAA compliant systems shall achieve an availability for A and Ai as listed in Table 2-7.

$$\begin{aligned} \text{Availability} = A &= (\text{Uptime})/(\text{Total Time}) \\ &= (\text{Operating Time} + \text{Standby}) \\ &\quad /(\text{Total Mission Time} - \text{Dormancy Time}) \end{aligned}$$

$$\text{Inherent Availability} = A_i = \text{MTBF}/(\text{MTBF} + \text{MTTR})$$

where:

Total Mission Time = Up Time + Down Time

Down Time = Mean Time To Repair (MTTR)

Dormancy Time is the time the system is in the Off Time, except possibly for low level of monitoring

MTBF is Mean Time Between Failure

MTTR is Mean Time To Repair



The actual values that are selected to replace the TBDs in the table are a design requirement, not an SGOAA requirement, and must be based on mission needs.

Table 2-7. Availability Requirements

FUNCTION	AVAILABILITY		CONFIDENCE FACTOR
	(A)	(AI)	
FC1	TBD	TBD	99%
MC2	TBD	TBD	95%
SC1	TBD	TBD	99%
NC3	TBD	TBD	95%
Overall System	TBD	TBD	95%

#### 2.2.4.2.5 Maintainability

SGOAA compliant systems shall support maintainability during in-space checkout, mission operation and on planetary surfaces, as required.

SGOAA compliant systems shall support achieving a Mean Corrective Maintenance Time (MCMT) of no greater than TBD minutes (EVA) and TBD minutes (IVA).

The Mean Time To Restore system for SGOAA compliant systems shall not exceed TBD minutes following a system failure.

As a minimum, the design of an SGOAA compliant system shall provide for modularity, accessibility and BIT to enhance installation simplicity and ease of maintenance.



## 2.3 SPACE GENERIC OPEN AVIONICS ARCHITECTURE DETAILED REQUIREMENTS DESCRIPTION

The SGOAA is based on partitioning between logical and direct requirements as illustrated in Figure 2-9. The model is established to include architectural functions, hardware, software and interfaces for all avionics systems. This SGOAA requirements description includes both system service software and applications software for the Space Data and Operations Control Subsystems. Interfaces in this model are valid for both one platform and multi-platform architectures on one or more vehicles. The SGOAA includes both processing service software and applications software for the space data and operations control subsystems as described in more depth later.

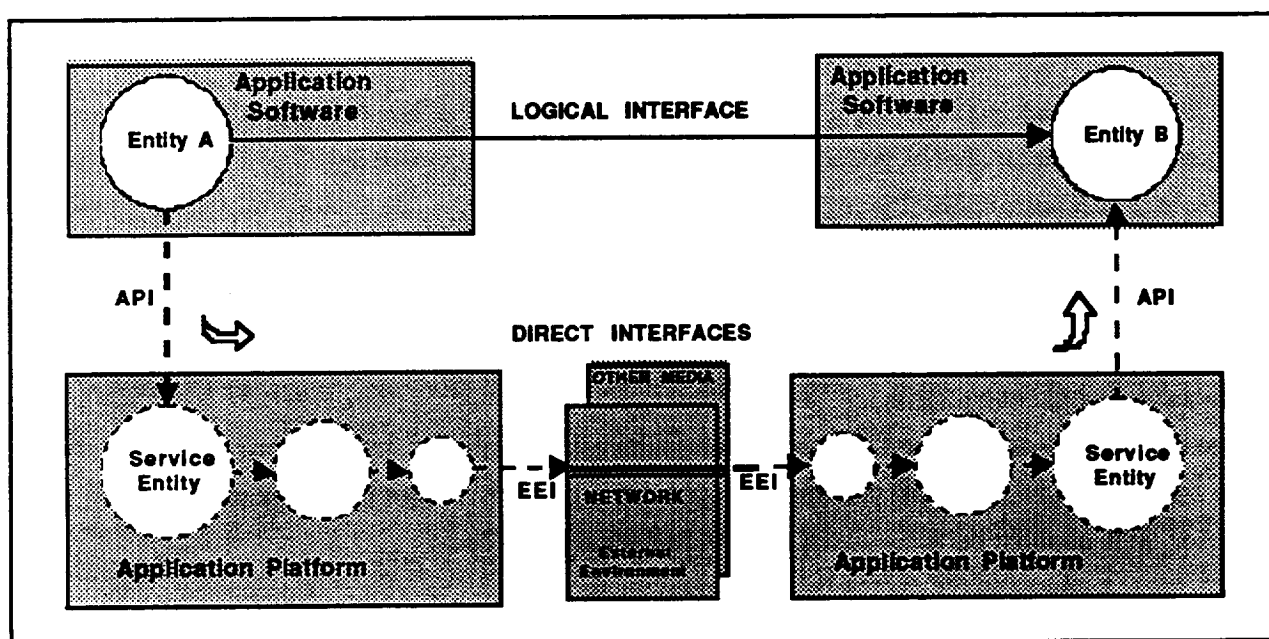


Figure 2-9. Logical System Requirements Flowdown to Direct Design Requirements

This model shall be used to define how system requirements are to be applied at the appropriate system level to determine the logical and direct interface points. System logical data flow requirements shall be created for each client/server entity addressing the data attributes needed by that entity or needed to be provided for some other entity. The logical data flow requirements shall identify the source of the data and the end-user needing the data, as well as the characteristic attributes required of the data. Logical data flow requirements shall not be concerned with the mechanism for implementing the data interchange. Implementation related requirements for the interfaces are a direct interface issue relating to the mechanisms provided for flowing the data from the source to the end-

user. Sources of the design requirements for the interfaces, application platform hardware and application platform services shall be derived from the Applications Software requirements and their logical data attribute requirements based on the user's needs.

### **2.3.1 GENERIC SYSTEM ARCHITECTURE REQUIREMENTS DESCRIPTION**

The SGOAA System Architecture Model shown in Figure 2-10 forms the basis for creating a model of the system under development. The generic and open system architecture proposed consists of processors which are standard, processors which can be tailored to users applications and missions needs, multiple communications mechanisms, and specialized hardware operating over standardized interfaces to the processors which manipulate the data they receive or provide.

System architecture models shall consist of a functional definition of the types of processors and communications paths required. The model shown in Figure 2-10 has three types of processors interconnected by two types of communications. This model only shows one of each type of hardware; the number of instances of each type of processor is variable depending upon system unique requirements and may range for 0 to n. For example, a centralized system architecture may look just like Figure 2-10, while a distributed system architecture may have multiple General Avionics Processor (GAPs), Special Avionics Processor (SAPs) and Embedded Processor (EPs). Either type of architecture may have many core networks and/or local communications mechanisms. More than one sensor and effector will usually be the rule in most non-trivial systems.

The processors shown in the system architecture in Figure 2-10 are a GAP for general purpose processing, a SAP for specialized processing support (vector/massively parallel/other), and an EP for the function of processing data within the sensor and effector devices. The sensors and effectors shown in the example may also interact directly with the main processors (the GAPs) or indirectly through EPs built into the sensors and effectors (if applicable).

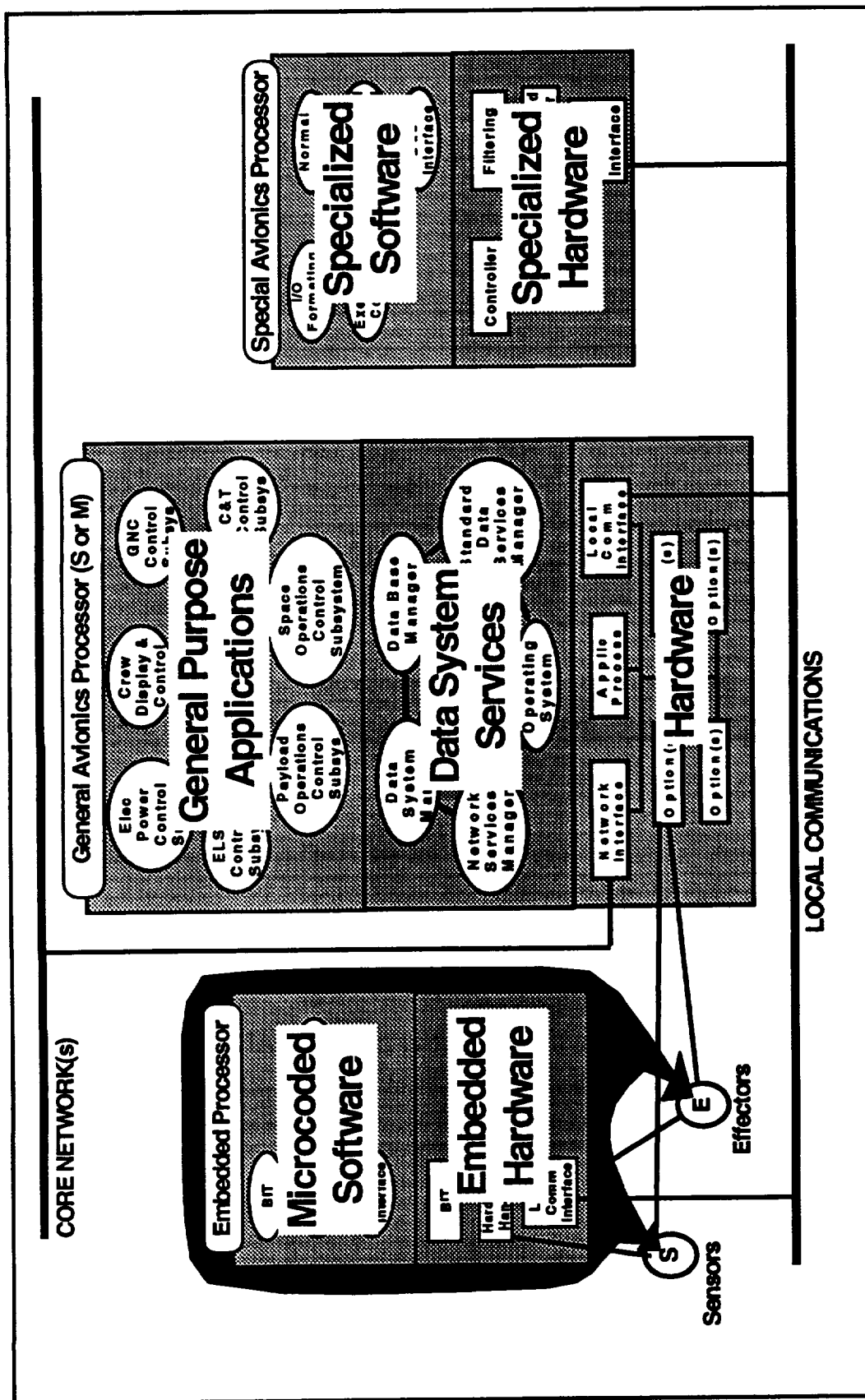


Figure 2-10. System Architecture Model

Communications paths illustrated are of two types: core networks for interconnecting sets of general processors or nodes, and local communications for interconnecting EPs and SAPs with their supported GAPs and general purpose processing applications. System models shall follow the general format of Figure 2-10, but shall be tailored to match individual system requirements.

There are sensors and effectors which can either interact directly with the main processors (the GAPs) or indirectly through the EPs built into the sensors and effectors (if applicable).

### **2.3.2 ARCHITECTURE INTERFACE MODEL REQUIREMENTS DESCRIPTION**

An architecture compliant with the SGOAA Interface Model requirements shall consist of six classes of interfaces as defined in Table 2-8. These classes are the levels of interfaces from hardware up to high level systems which shall be completely defined in an architecture developed in accordance with this standard. Definition of each interface class shall be in accordance with the requirements contained in the following paragraphs.

The relationships of these interface classes to the POSIX Model is shown in Figure 2-11 and Section 2.4 provides an in-depth discussion of the SGOAA relationships to the POSIX Model.

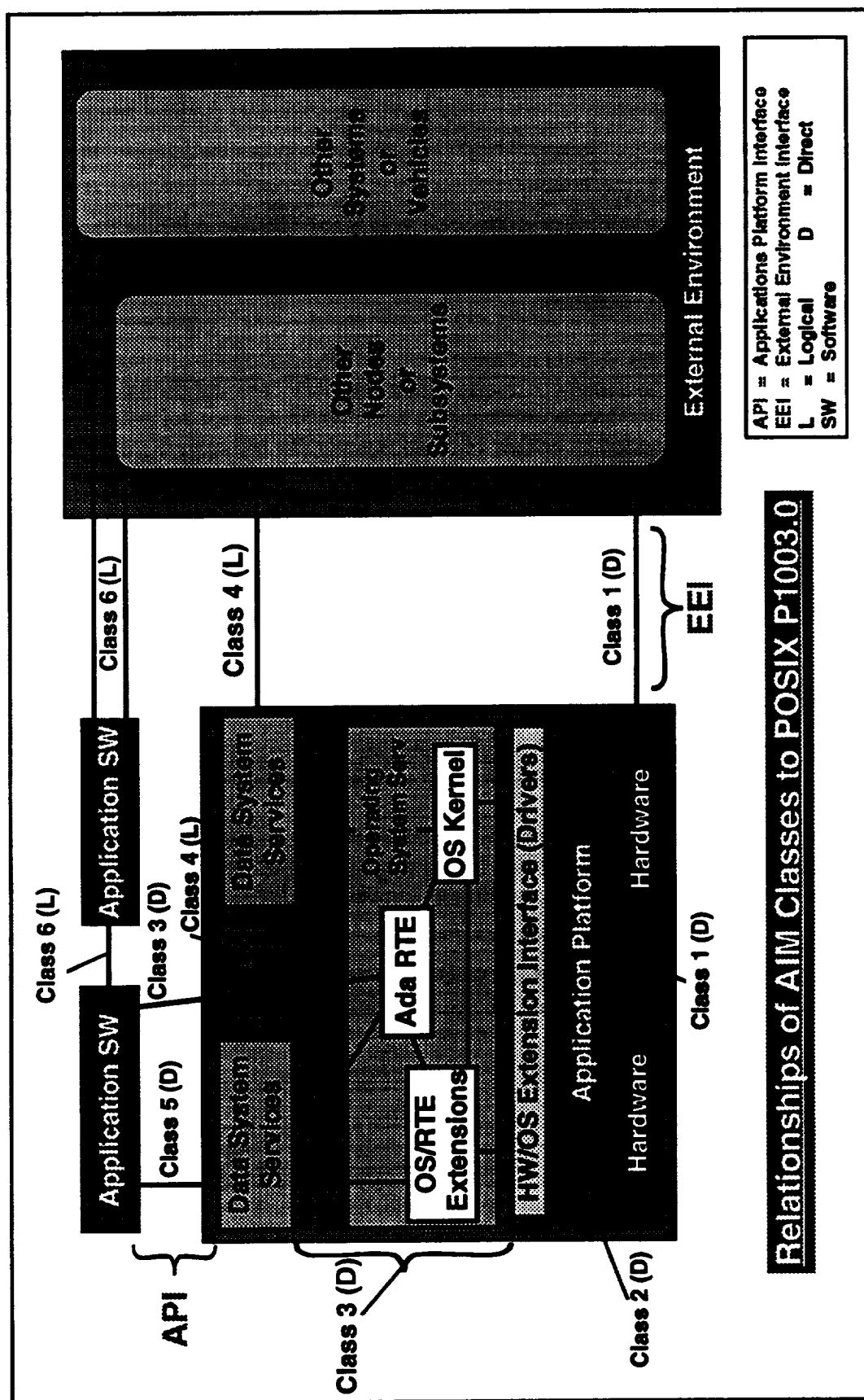
The following subsections describe each of these interfaces in more detail and provide examples to clarify the use of these interface standard classes.

#### **2.3.2.1 Class 1 - Hardware-to-Hardware Direct Interfaces**

The Class 1 Hardware-to-Hardware Direct Interface include three key aspects of the class 1 direct interface: the interface architecture, the generic processing external hardware architecture, and the general avionics processor internal hardware architecture.

Table 2-8. Architectural Interface Classes

CLASS	DESCRIPTION
1	<p><b>Hardware-to-Hardware Direct:</b>  Class 1 hardware direct interfaces are the direct connections between different types of hardware such as needed to enable buses and communications links to address processors or needed to enable processors to address memory registers.</p>
2	<p><b>Hardware-to-System Software Direct:</b>  Class 2 hardware to system software direct interfaces are the direct connections between hardware registers and system service software drivers, such as needed to enable address registers to move data packets from hardware to system service software, and service drivers which can respond to the data packets.</p>
3	<p><b>System Software-to-Software (Local) Direct:</b>  Class 3 system service software to other software direct interfaces are the direct connections between operating system service code and other local software code sets, which enable operating system software to receive and interpret data packets, and pass them on to other software code which will process them locally.</p>
4	<p><b>System Software-to-System Software Logical:</b>  Class 4 system service software to other system service software logical interfaces are the indirect connections which enable local service software to determine the address of the intended software in other local or remote locations which need the register data being stored and to pass the data appropriately. Enables the handling of logical data transfers from source to user service</p>
5	<p><b>System Software-to-Applications Software Direct:</b>  Class 5 system service software to applications software direct interfaces are the direct connections which enable software service code to access and process data from local application software code.</p>
6	<p><b>Applications Software-to-Applications Software Logical:</b>  Class 6 applications software to applications software logical interfaces are the indirect connections which enable an application originating data to pass it to an application which needs to use the data, or enable an application needing data to determine the source from which the data must be obtained. These are logical data transfers from source to user. This interface provides the indirect connections that allow applications in different systems or in the same system to communicate, thus enabling applications software to interact across system boundaries or within system boundaries to accomplish a mutual purpose. These interfaces may be applicable to applications executing in the same processor, in different processors in the same node or in different systems.</p>





#### **2.3.2.1.1 Hardware Interface Architecture Models**

Hardware to hardware direct interfaces are shown in Figure 2-12. These interfaces consists of the nuts, and bolts, chips and wires of the hardware architecture described previously. With regard to the model, this interface consists of all the hardware to hardware interfaces within each processing element, as well as the hardware interfaces to the external environment by way of the core network, local communications or direct interfaces. This architecture provides for three classes of processors: the EPs, SAPs and GAPs for which standardized interfaces are required to be selected from a set of acceptable lower level interface standards.

The GAP architecture can be configured to provide hardware components to interface to a core network, to interface to local buses, to process applications, and optional components for other purposes (such as serial input and output to direct analog and discrete links). The SAP architecture can be configured to provide hardware components for control, filtering, bus interface and other specialized purposes. The EP architecture can be configured to provide hardware components for microcontrol, BIT, hardware handling and setup, and bus interface. As shown in Figure 2-12, the GAP is the focus of efforts to standardize the hardware processor support due to its general purpose nature.

This architecture can also be configured to provide communications capabilities from three classes of communications: local communications within a subsystem between multiple processors, core network communications between multiple subsystems or systems and direct communications to embedded processors to sensors and effectors. The local communications can be implemented by a combinations of buses and direct links for analog, discrete or serial communications between subsystem elements or components. Core networks can be implemented by high capacity buses such as FDDI or by direct links between high data rate elements. The communications from sensors or effectors to EPs are only possible through direct links because the intention of the architecture is that embedded processors are those processors embedded in the sensor or effector hardware devices to minimize the communications latencies since some of the sensors and effectors will have very high data rates and very low tolerance to latency or time delays.

# Hardware Application Platform

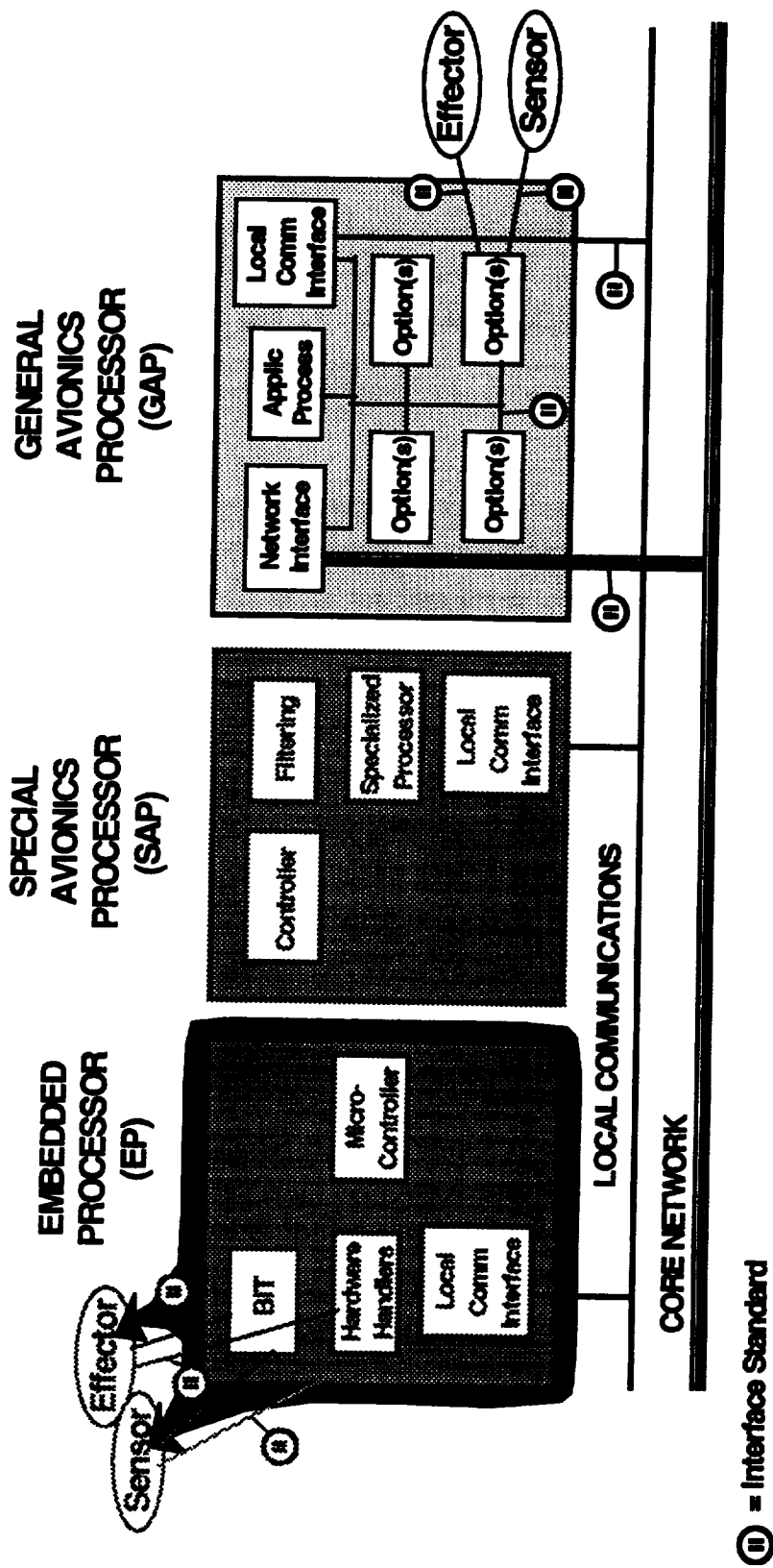


Figure 2-12. Class 1 Hardware to Hardware Direct Interfaces

The external GAP hardware interface standards needed are identified in Figure 2-13 for this architecture. The interfaces are shown in black, and everything else has been greyed out of focus (note that the resulting GAP internal architecture is shown in full, without greying, in Figure 2-16).

#### **2.3.2.1.2 Generic Processing External Hardware Architecture Models**

The System Hardware Architecture is shown in Figure 2-14. The architecture core network represents the inter-subsystem connectivity, and can be implemented by a combination of one or more communications paths using point-to-point, ring, bus or other architecture designs. Typically, core networks are implemented by lower level standards such as Fiber Data Distribution Interface (FDDI) or Ethernet. Local communications provide the intra-subsystem connectivity for high speed data communications between processors within one subsystem. Typically, the local communications are implemented by lower level standards such as MIL-STD 1553B for local command and telemetry data buses, RS-488 for timing controls, and direct links for analog and discrete signals. The interface plugs shown represent the unique hardware interfaces which must be defined by standards and handled in processing.

GAPs represent the general purpose processing used by the embedded computer systems. GAPs are allowed to be of two forms: one for standard general purpose use [GAP(S)] and one for multiplexing and demultiplexing signals [GAP(M)]. Typically, GAP devices are used where response times on the order of seconds to tens of seconds are required. An example of a compliant implementation of GAP(S) processors is the Standard Data Processor (SDP) in the Space Station Freedom program and the General Purpose Processing Element in the F-22 program. An example of a compliant implementation of the GAP(M) is the Multiplex-DeMultiplex (MDM) processor in the Space Station Freedom program.

SAPs represent the special purpose processing which is usually needed in high power embedded computers; these could be implemented by devices such as vector or associative processors, massively parallel data processors, or arithmetic coprocessors. Typically, SAP devices are used where response times on the order of hundreds of milliseconds to a second are required. Examples include the associative and vector processors used in the F-22 program.

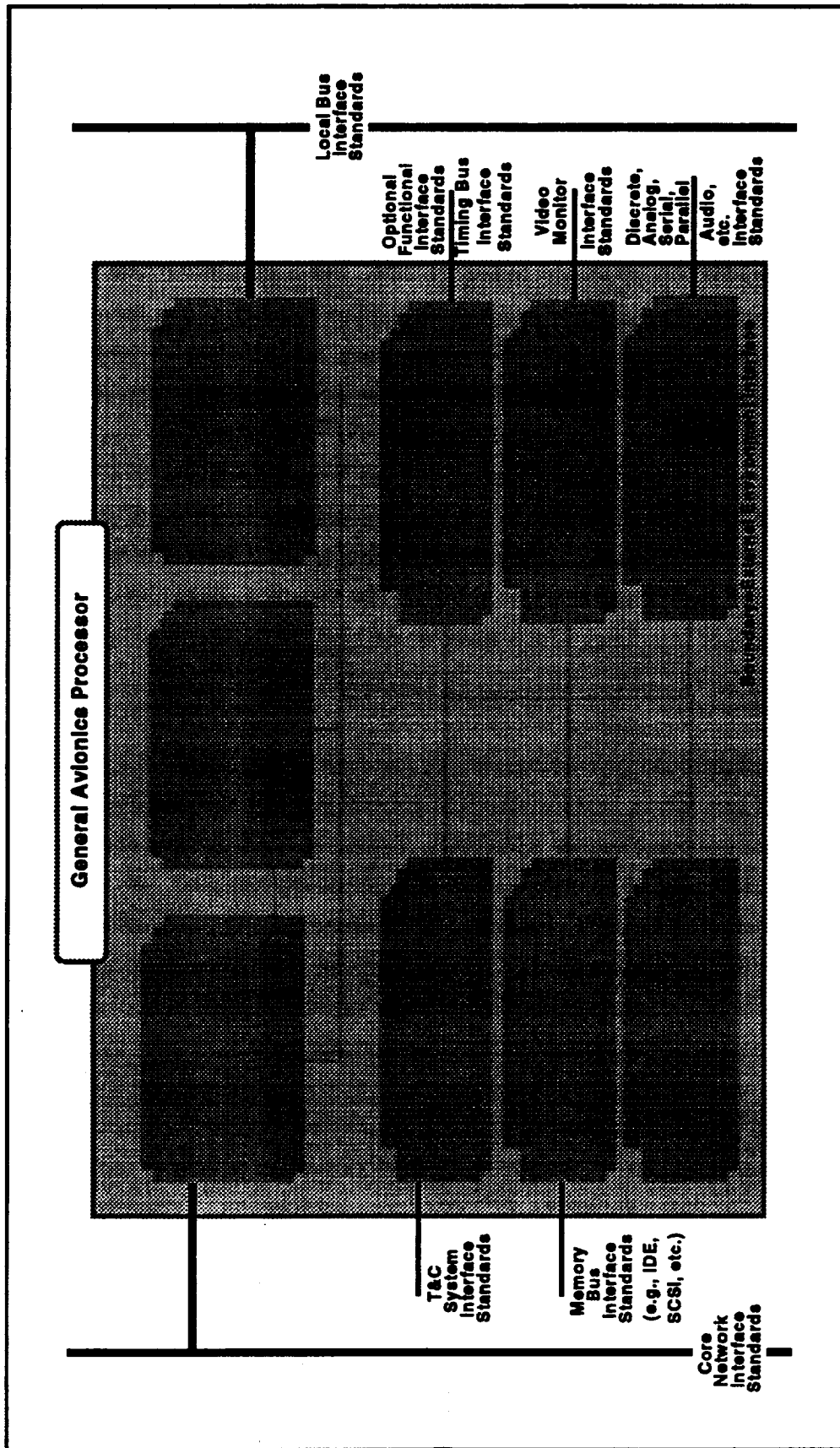


Figure 2-13. GAP Hardware to Hardware Interface Standards

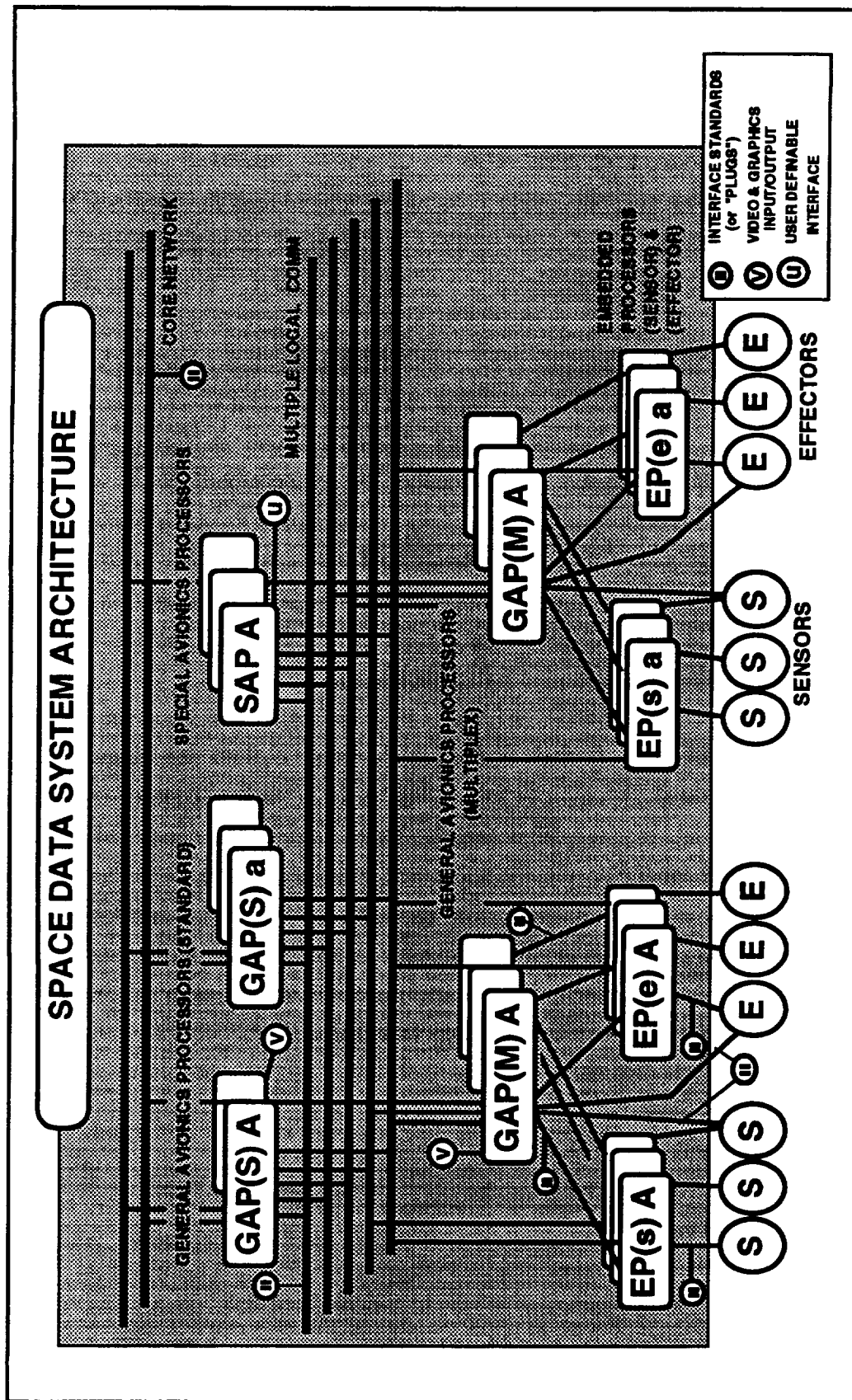


Figure 2-14. Generic Processing External Hardware Architecture and Interfaces for a Space Generic Open Avionics Architecture

Within each sensor or effector, this architecture allows, but does not require, the placement of processors embedded in the sensor or effector unit. An EP can be one of two forms: one for effector processing [EP(e)] and one for sensor processing [EP(s)]. These embedded processors provide the very high speed processing to manipulate and convert analog data to digital data while performing some preprocessing on it to reduce the data rate to a more acceptable level for linkage back to the GAP(M). Typically, EP devices are used where response times on the order of milliseconds or less are required. Where the data rate with the sensor or effector is acceptable to the GAP(M) and no other pre-processing is required, direct interface to the GAP(M) may be used. Sensors and effectors interface to the EP devices either through local communication interfaces or through direct links.

Lower level interface standards are used to define the options available in implementation for the core networks, local communications, GAP to EP direct links, GAP to S direct links, GAP to E direct links, EP to S direct links, and EP to E direct links. User definable interfaces are provided for the SAPs. Lower level video and graphics interface standards are used to define the options available in implementation for connecting the GAP devices to humans for development, operation and maintenance of the systems.

The standard system architecture such as used in the Space Station can be overlaid with the POSIX OSE interfaces, as shown by example in Figure 2-15. This is another way of looking at the use of the OSE model. SDPs, MDMs and the Sensor and Effector Embedded Processors are the host computers for the Application Platform and its services, as well as the Application Software. Communications from the SDPs over the core network, local buses and direct communications links are communications to other standard processing elements, hence are external interfaces. Communications within each processor (whether the SDP, MDM SP or the EP) is an internal interface (the API).

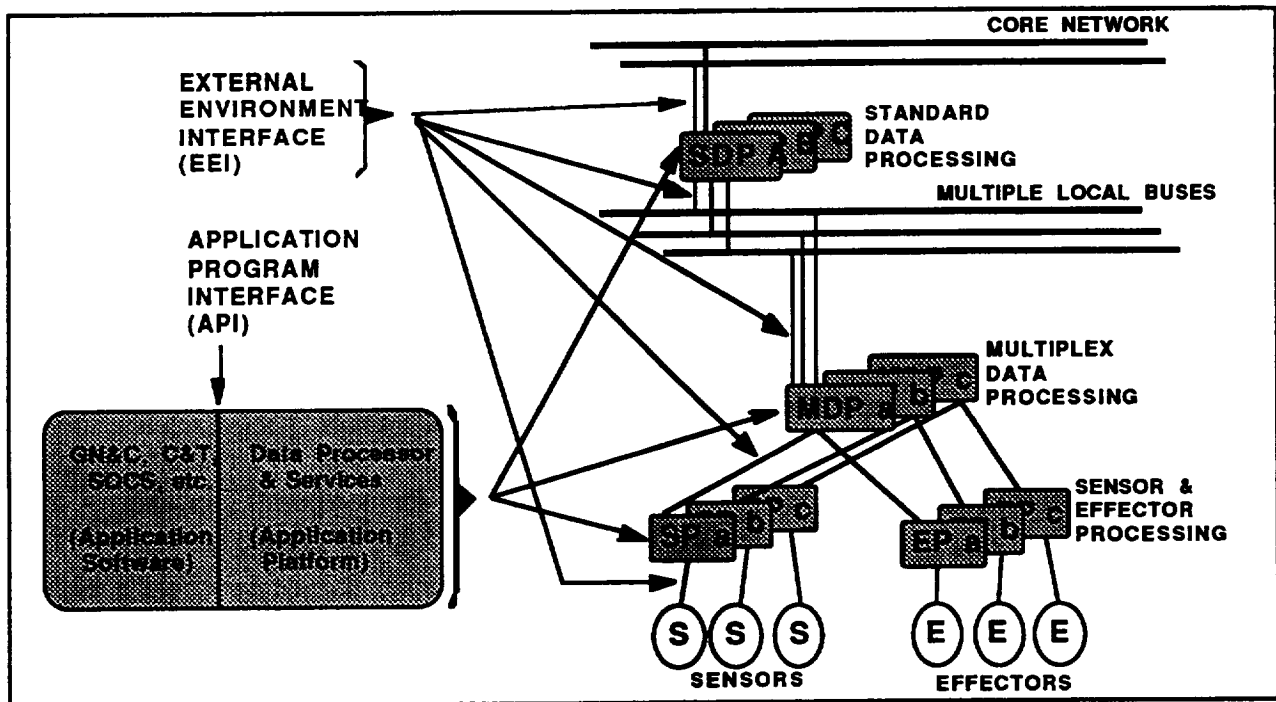


Figure 2-15. POSIX Open System Environment Interfaces Applied to a Station Example of a Standard Hardware Architecture

### **2.3.2.1.3 Generic Processing Internal Hardware Architecture Model**

Figure 2-16 presents a GAP architecture that is capable of being configured to satisfy the requirements for general purpose processing elements in a spacecraft. The generic hardware elements shown in the figure comprise the basic, generic hardware modular elements in the SGOAA. The processor may be configured as a GAP(S) or GAP(M) as illustrated in Figure 2-14 depending on the set of available functions required for the specific application. The SAP is a special purpose case and may require functions not included in the generic processor function set such as vector or parallel processing.

The GAP function set is a shopping list of modular functions which can be used to build the needed configuration. Each module provides a specific independently procurable service. Additional unique service functions may be added by defining additional modules. The actual implementation in hardware is interface standard, technology and detailed design dependant. System performance requirements for hardware modular elements shall be a primary consideration in module selection to perform a specific function.

System Fault Detection, Isolation and Recovery requirements for hardware modular element BITE shall also be considered in hardware modular element selection. Specific hardware interfaces that are candidates for standards are shown in the figure. If modules interact, specification of the interface between modules may be standardized or non-standard. Interfaces between modules are processor internal interfaces. Standardization of the internal interfaces provides portability and interoperability of the processor modules. Certain interfaces between modules for special processing functions may be quite specialized, complex and varied and as such may not be standardized.

Backplane bus interface standards shall be imposed to provide modularity with the capability for technology upgrades and multiple vendor sources of processing functions modules. Although only one bus is shown for the backplane in Figure 2-16, the actual bus implementation may consist of multiple buses depending upon the specific application. Possible buses include data, time, test, and local memory. Multiple standards exist for all of these bus types. Bus interface standards provide for cost and schedule savings by using a predefined standard interface as well as making common bus interface hardware useable in all system processing elements. Use of Backplane Bus Interface Standards first requires identification of the backplane bus function to be performed; in this case, the function shown is the connection of processing elements across the backplane of a single board



computer. Similar type savings can be achieved by implementing standards for the other interfaces labeled in Figure 2-16.

Lower level interface standards should be selected for network, test and checkout system, mass memory, timing bus, discrete data, analog data, serial data, parallel data, local bus, video/graphics, audio and optional functional growth interfaces. For example, to implement the functions of the basic GAP(S) shown in Figures 2-13 and 2-14 would require implementation of the network processing, application processing and local communications (e.g., buses and I/O) processing functions of the GAP Hardware Architecture shown in Figure 2-16. A backplane bus standard such as Future Bus Plus (FB+), VME or Pi Bus would be imposed as the backplane data bus standard. The backplane bus standard used in a specific architecture implementation might consist of one or more specific buses; separate buses are permitted for uses such as test and maintenance.

#### **2.3.2.2 Class 2 - Hardware-to-System Software Direct Interfaces**

Hardware to system software interfaces are shown in Figure 2-17. These interfaces consist of the interfaces from the system software drivers (i.e. in the OS, data system manager, etc.) to the hardware instruction set architecture (ISA) and register usage. With regard to the model it is internal to each processing element. The hardware elements are grayed out to show that these elements are a repeat of the previous figure; the black elements represent the new capabilities and interfaces added by this interface class. This class defines the interfaces for low level software drivers that interact with the hardware for each of the processor types (EPs, SAPs, and GAPs). The drivers are hardware dependent, but this enables the architecture to begin to partition out the hardware dependencies, which is a key in providing for technology upgradability in the future. All the drivers for all processor types are contained in the SDSS sub-architecture.

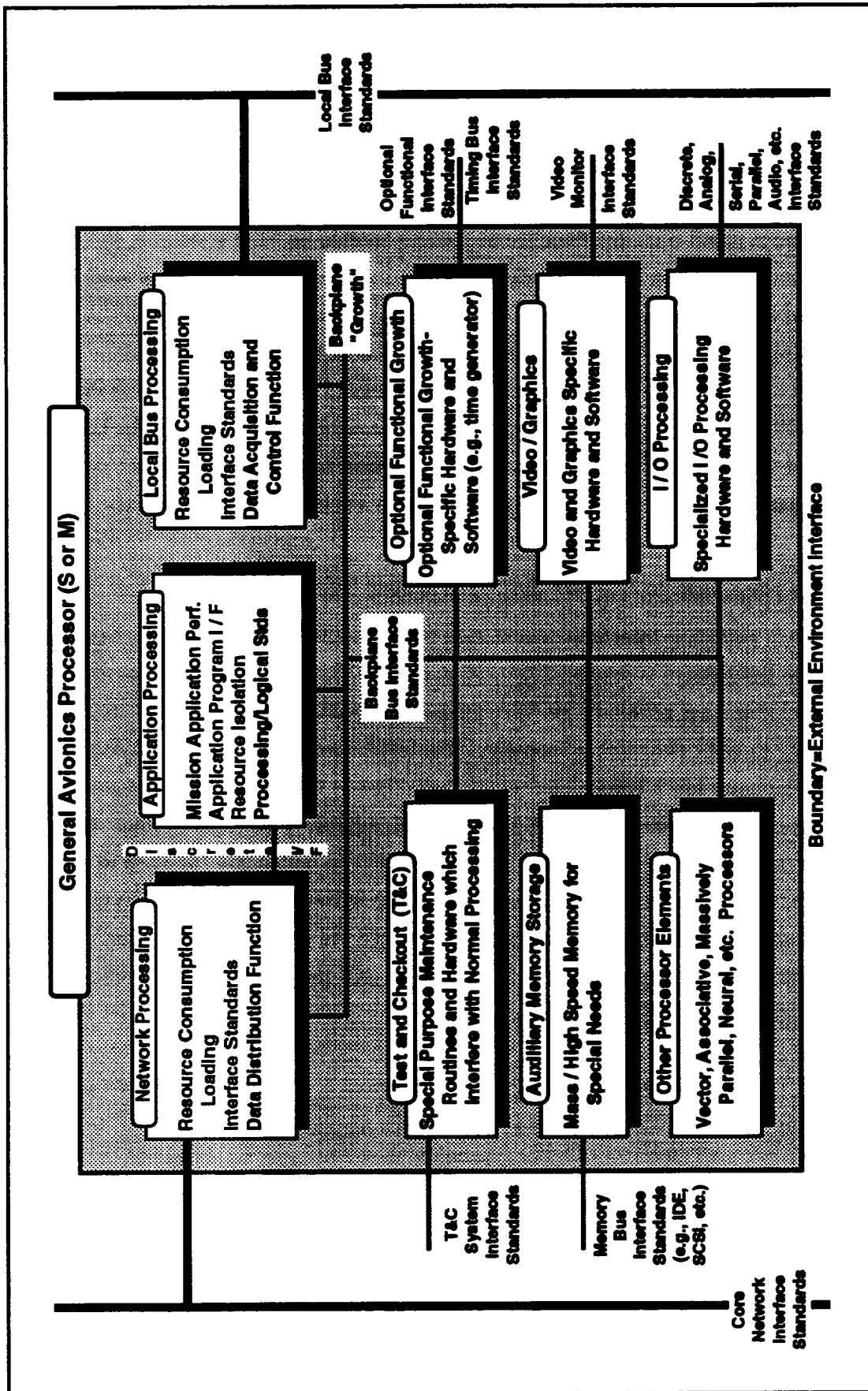


Figure 2-16 Generic Processing Internal Hardware Architecture Model

The system services software for the GAP are organized into five categories, as discussed in more detail in section 3. These categories are the Data System Manager, Data Base Manager, Standard Data Services Manager, Operating System, and Network Services Manager. The software drivers for the SAP are organized into four categories (preliminary): input/output (I/O) formatting, normalization, specialized processing interfaces, and local communications interfaces. The software drivers for the EP are organized into four categories (preliminary): BIT, hardware handler interfaces, local communications interfaces and microprocessor execution control. Note the naming convention between Figures 2-17 and 2-18. Interfaces identified in Figure 2-17 are labeled with a name (e.g., GAP-DRVR for the GAP hardware to service drivers), and then these named interfaces are exploded in the next figure (e.g., GAP-DRVR-OP, GAP-DRVR-MEM, GAP-DRVR-TC, etc) by adding a third name to the first two which identifies the component driver of the interface.

The operating system interfaces needed for the hardware to drivers are identified in Figure 2-18. The interfaces are shown in black and labeled, and everything else has been greyed out to highlight items of interest.

#### **2.3.2.3 Class 3 - System Software-to-Software (Local) Direct Interfaces**

System software to local system software service direct interfaces are the operating system interfaces shown in Figure 2-19. These consist of the Input/Output handler calling conventions and context switch conversions between the system software drivers on one processing element interfacing with one or more system software services to provide for local information exchange. The grayed out parts of the figure represent the material covered in Classes 1 and 2, the black parts of the figure are the new material added in Class 3. Since Class 2 provided the software drivers to isolate the hardware, Class 3 provides the remainder of the direct operating system interfaces to local software services needed to operate the computer system. All local software services are grouped into the SDSS sub-architecture, consisting of the Data System Manager, Data Base Manager, Standard Data Services Manager, Operating System, and Network Services Manager. Class 3 provides the direct interface between the local services for effective local interprocess communications and support. These interfaces are direct interfaces because they enable software service code to interact with software service code in other local entities. Class 3 interfaces meet derived requirements based on the need of an application to support users.

# Application Platform Hardware to Service Drivers

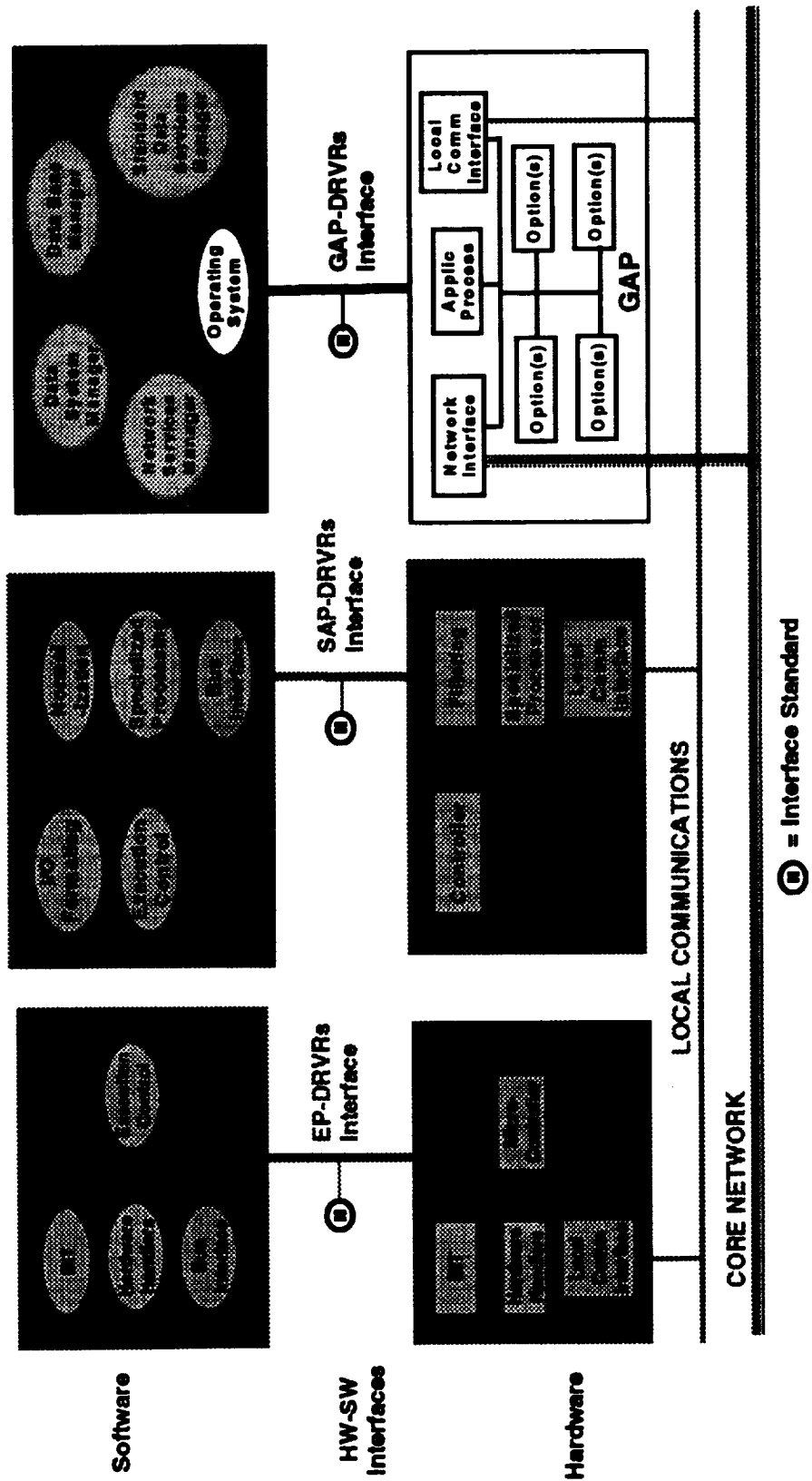


Figure 2-17. Class 2 Hardware-to-System Software Direct

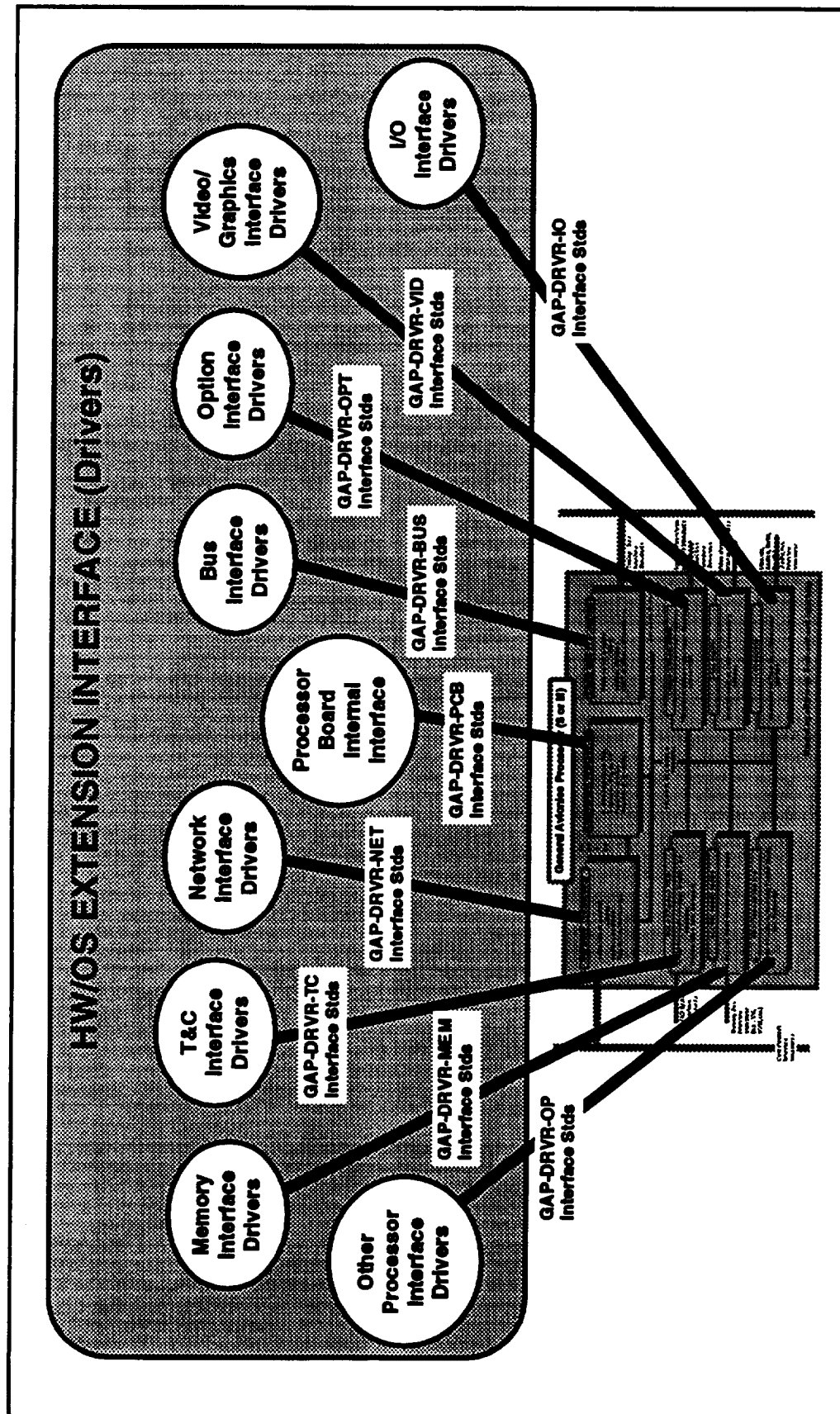


Figure 2-18. GAP to Operating System Hardware Drivers

## Operating System to Other Code (i.e., Services or Applications)

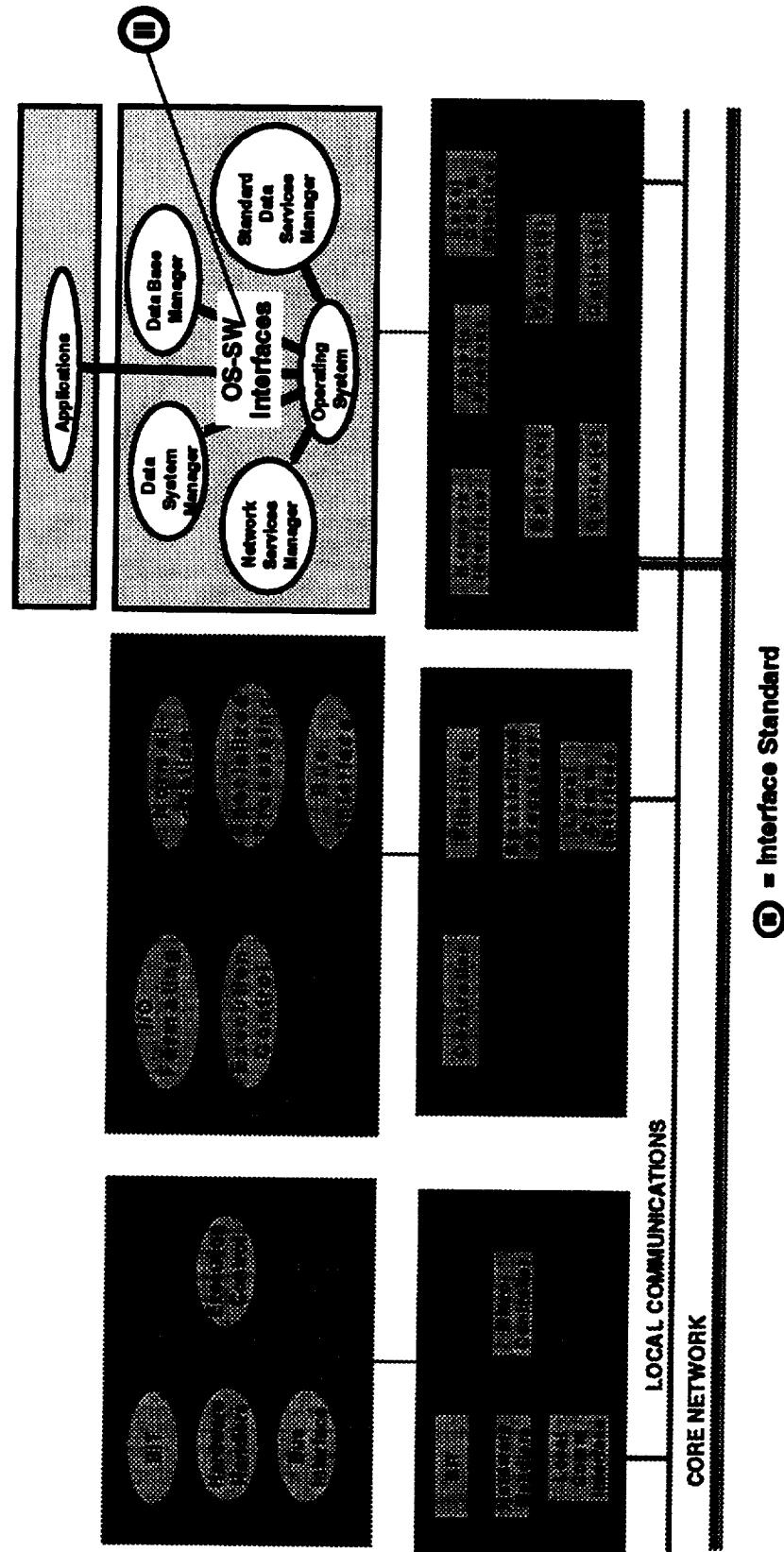


Figure 2-19. Class 3 System Software-to-Software Direct Interfaces

The operating system interfaces needed to implement system software-to-system software direct interfaces are identified in Figure 2-20. The interfaces are shown in black and labeled, and everything else has been greyed out to highlight items of interest. Note that there are two types of interfaces: upward between the operating system services to any software application (including other data system services), and downward to the drivers within the operating system. The naming conventions (previously described) of using two concatenated names for the higher named interface in the first figure, and three concatenated names for the explosion path in the second figure, are also applied here.

#### **2.3.2.4 Class 4 - System Software-to-System Software Logical Interfaces**

System software services to remote system software interfaces are shown in Figure 2-21. This is the peer to peer interface of system software in one processing element (GAP,SAP or EP) interfacing with the system software in the same processing element or in an external processing element to coordinate operations in a distributed environment. The grayed out parts of the figure represent the material covered in Classes 1 to 3, the black parts of the figure are the new material added in Class 4. Since Classes 1 to 3 isolated the hardware and software services in each processor, Class 4 adds the interface capability for services in one processor to interact with services in another processor; this is the heart of multi-processor capability needed in modern space avionics systems. EP services can interact with SAP and GAP services; SAP services can interact with GAP services; GAP services can interact with EP and SAP services and other GAP services. These interfaces are logical interfaces because the service originating data is interacting with the service that will use the data (i.e., that will transform the data into another form for a purpose). Class 4 interfaces meet derived requirements based on the need of an application to support users in a multi-processing environment.

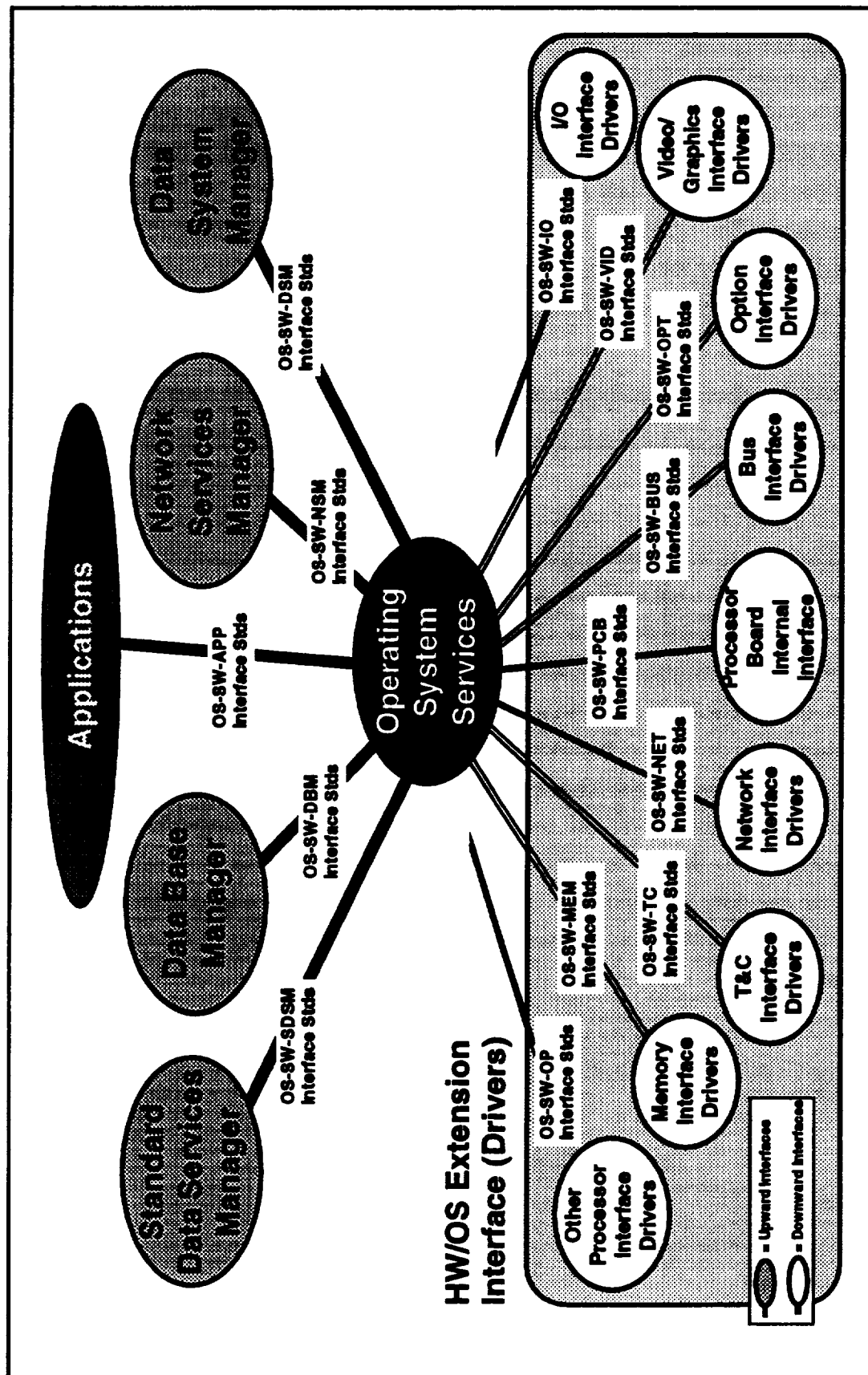


Figure 2-20. Operating System Interfaces





The data system services interfaces needed to support local operations and logical access to other GAP data system services are identified in Figure 2-22. Although this diagram is busy, a little study will reveal the underlying pattern. The black-line interfaces are the primary interfaces between the local services. Local services and remote services have a common logical architecture. Also, shown in Figure 2-22 is a circular interface between each service entity and itself, since each service must be able to communicate with remote versions of itself in other nodes. Finally, there are remote interfaces to the special avionics processor and the embedded processor services not illustrated in this figure. The naming conventions (previously described) of using two concatenated names for the higher named interface in the first figure, and three concatenated names for the explosion path in the second figure, are also applied here.

#### **2.3.2.5 Class 5 - System Software-to-Applications Software (Local) Direct Interfaces**

System software services to applications software interfaces are shown in Figure 2-23. This is the direct interface within a processing element between the application software and the system software (language bindings/specification) to allow provision of needed services. The grayed out parts of the figure represent the material covered in Classes 1 to 4, the black parts of the figure are the new material added in Class 5. Since Classes 1 to 4 isolated the hardware and software services in all the processors, Class 5 adds the interface capability for services in any processor to interact with an application executing in the processor. This provides the basic multi-processor capability to meet actual user requirements in processing. Applications can operate in any GAP, with potential partitioning of an application across multiple GAPs. Similarly, applications can operate in any SAP or any EP. These interfaces are direct interfaces because the applications software code is interacting with the service software code. Class 5 interfaces meet derived requirements based on the need of an application to support users in a multi-processing environment.

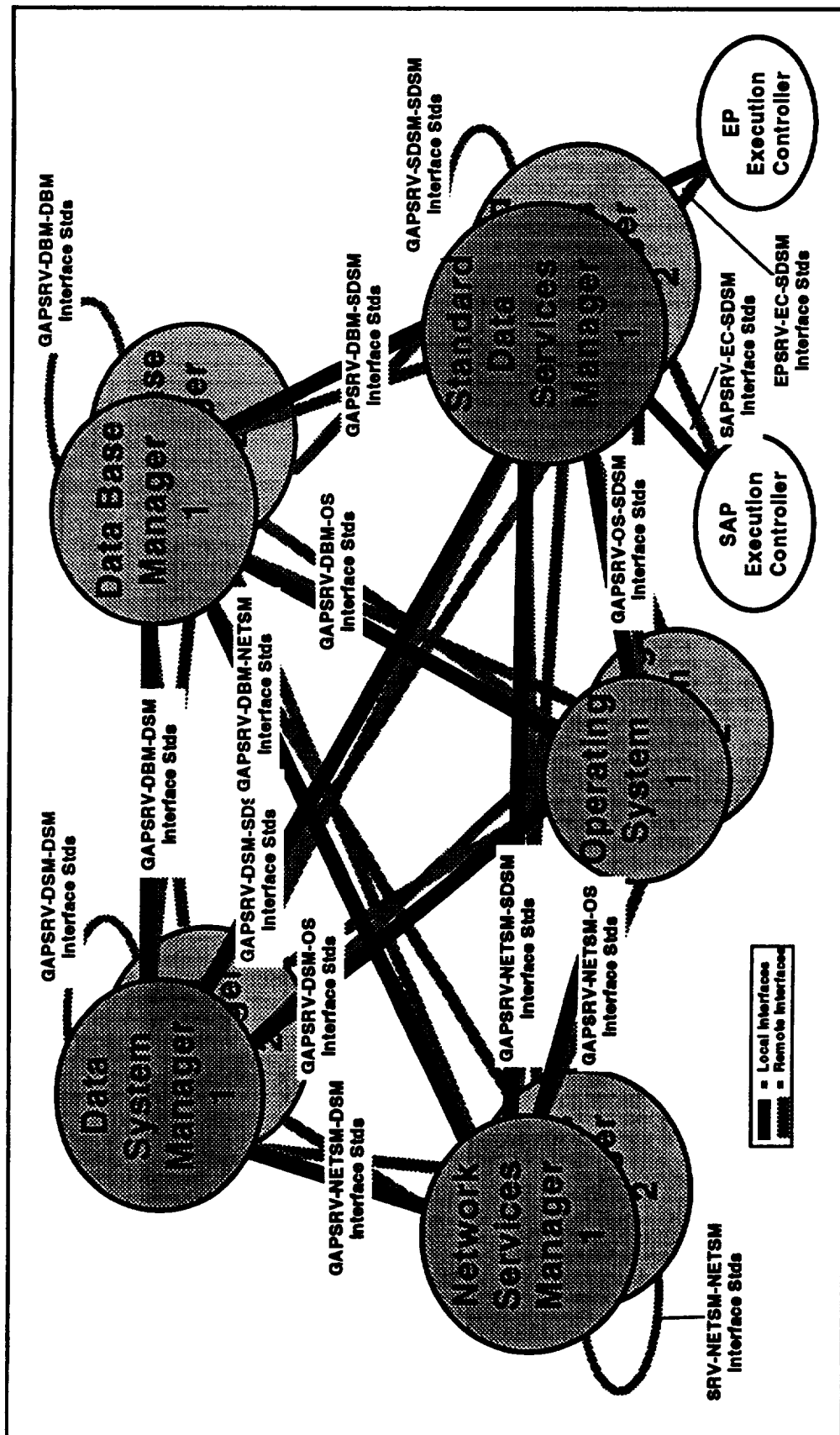


Figure 2-22. SDSS Services to Other or Remote Services

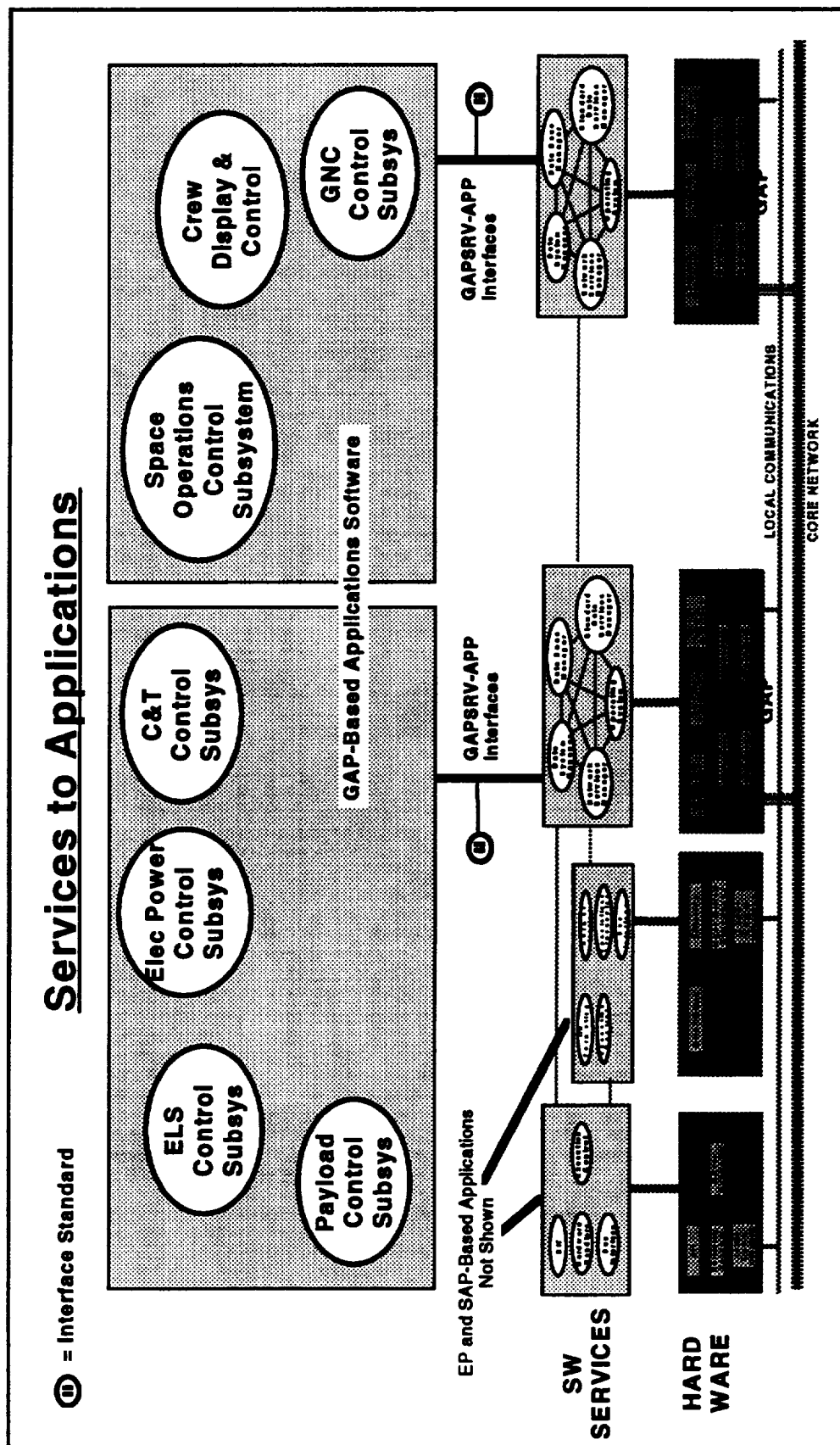


Figure 2-23. Class 5 System Software-to-Applications Software Direct Interfaces

The services to applications interfaces needed are identified in Figure 2-24. The interfaces are shown in black and labeled, and everything else has been greyed out to highlight items of interest. Note that all applications can be represented by one bubble since there should be a standardized method of access to the data system services which is a function of the service and is independent of any one application. The standard data services manager shall be capable of providing access to other services as well as directly to the application or sensor providing the source of data. The data system manager shall be capable of providing control interfaces to other control subsystems. The naming conventions (previously described) of using two concatenated names for the higher named interface in the first figure, and three concatenated names for the explosion path in the second figure, are also applied here.

#### **2.3.2.6 Class 6 - Applications Software-to-Applications Software Logical Interfaces**

Applications software to applications software interfaces are shown in Figure 2-25. This is a peer to peer information exchange and coordination interface between application software modules. Applications may not communicate directly. All application to application software communication must be implemented by use of system services software. All communication is through a Class 5(P) standard interface to System Services which provides the direct communications path between applications. This interface may be between applications within a processing element or between applications in separate processing elements. The grayed out parts of the figure represent the material covered in Classes 1 to 5, the black parts of the figure are the new material added in Class 6. Since Classes 1 to 5 isolated the hardware, software services and applications in any processor, Class 6 adds the interface capability for an application in any processor to interact with another application executing in any processor. Applications can operate in any processor (i.e., GAP, SAP or EP), with cooperating interactions to support the needs of the users. This interface is a logical interface to establish the requirements for information exchange from one application to another, i.e., the application originating data is interacting with applications that will use the data (i.e., that will transform the data into a form useful to the user or to another application for a user's ultimate purpose). Class 6 interfaces meet user and derived requirements based on the need of multiple applications to support users in a multi-system environment.

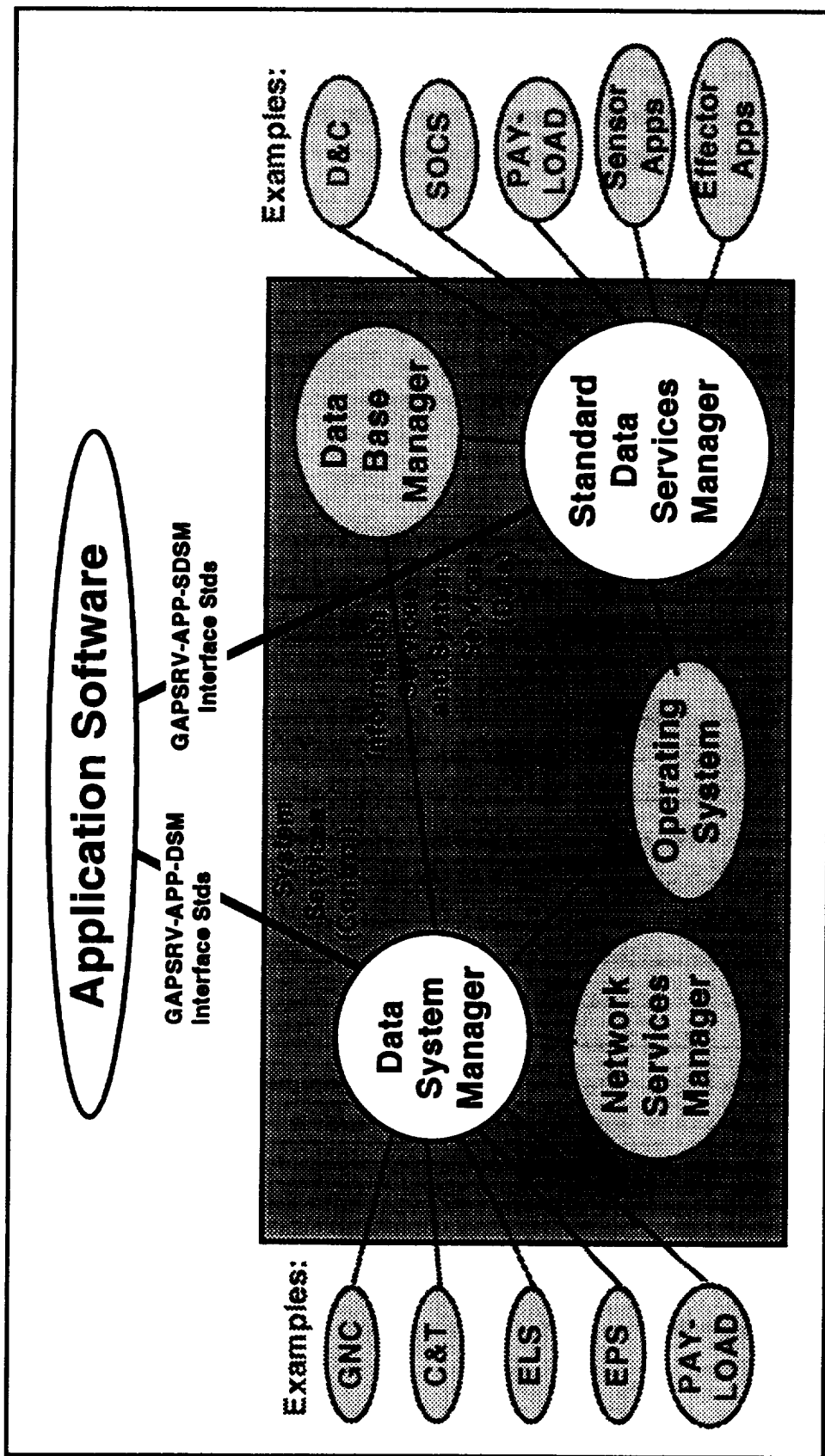


Figure 2-24. Services to Applications Interfaces

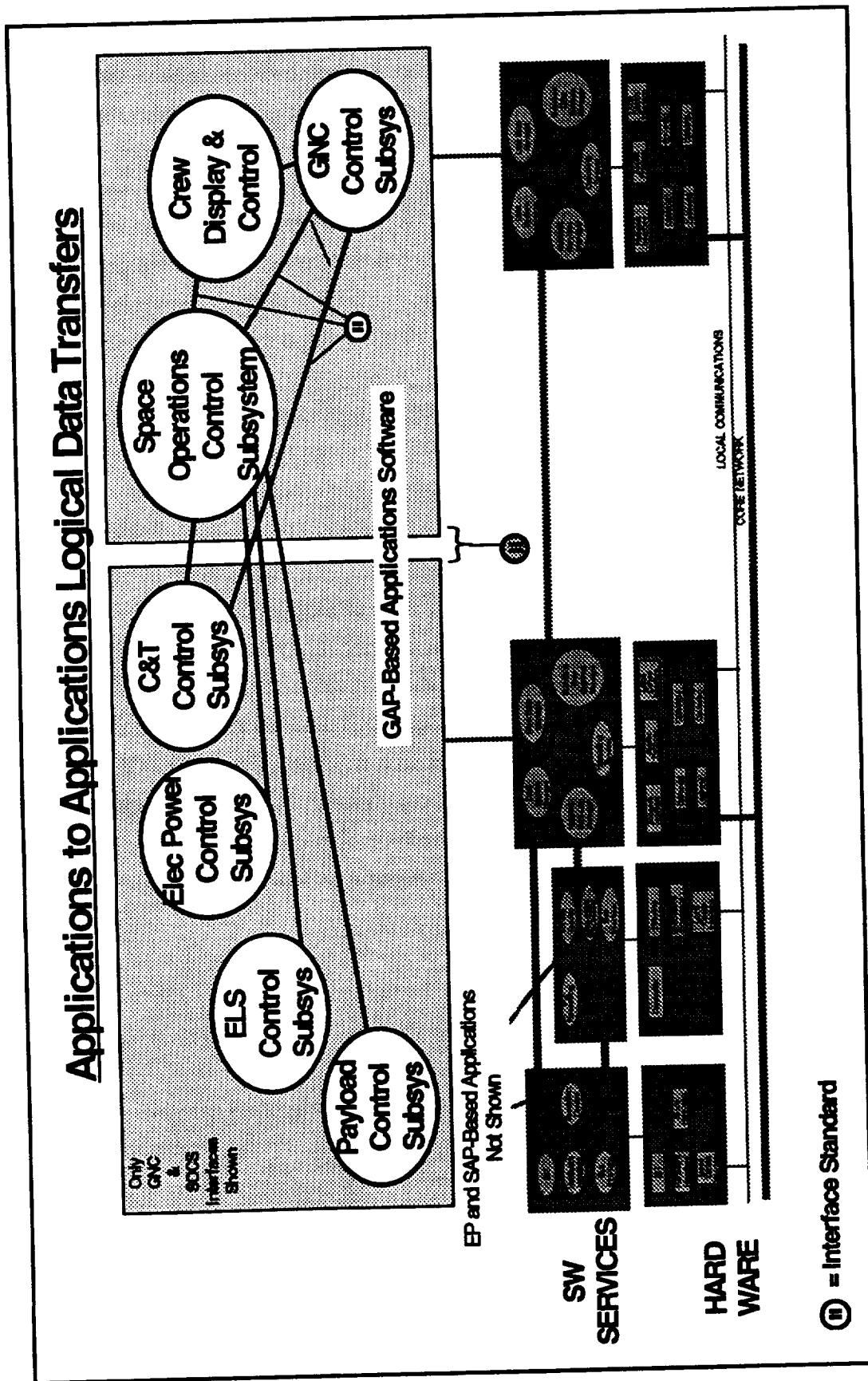


Figure 2-25. Class 6 Applications Software-to-Applications Software Logical Interfaces

Applications to Applications interfaces can also include interfaces between applications in two different systems or vehicles. Thus, System A applications software to system B applications software interfaces are shown in Figure 2-26. This is the interface for exchange of information between the space avionics system and another avionics system for overall command and control. This interface is at the mission level and may be an information exchange between the ground or between separate space vehicles. The grayed out parts of the figure represent the material covered in Classes 1 to 6 (within one system), the black parts of the figure are the unique material added to Class 6 for inter-system interfacing. Since Classes 1 to 5 isolated the hardware, software services and applications in any system, Class 6 adds the interface capability for an application in one system to interact with an application executing in another system. Class 6 interfaces shall meet user and derived requirements based on the need of multiple applications to support users in a multi-system environment, comprising multiple systems, facilities or vehicles. Applications can operate in any system's processor (e.g., the Mission Control Center GAP or workstation) to cooperate with applications in another system's processor (e.g., the Lunar Transfer Vehicle GAP). The interfaces are logical interfaces because the application originating data is interacting with applications that will use the data in another system, facility or vehicle (i.e., that will transform the data into a form useful to the user or to another application for a user's ultimate purpose). Class 6 interfaces meet user and derived requirements based on the need of multiple applications to support users in a multi-system environment. They meet the overall mission and operational control requirements across multiple facilities and vehicles. With regard to the hardware architecture, the communication gateway might be a SAP configured as an RF Communication Processor or a SAP configured as a Network Gateway.



# **System-to-System Applications Logical Data Transfers**

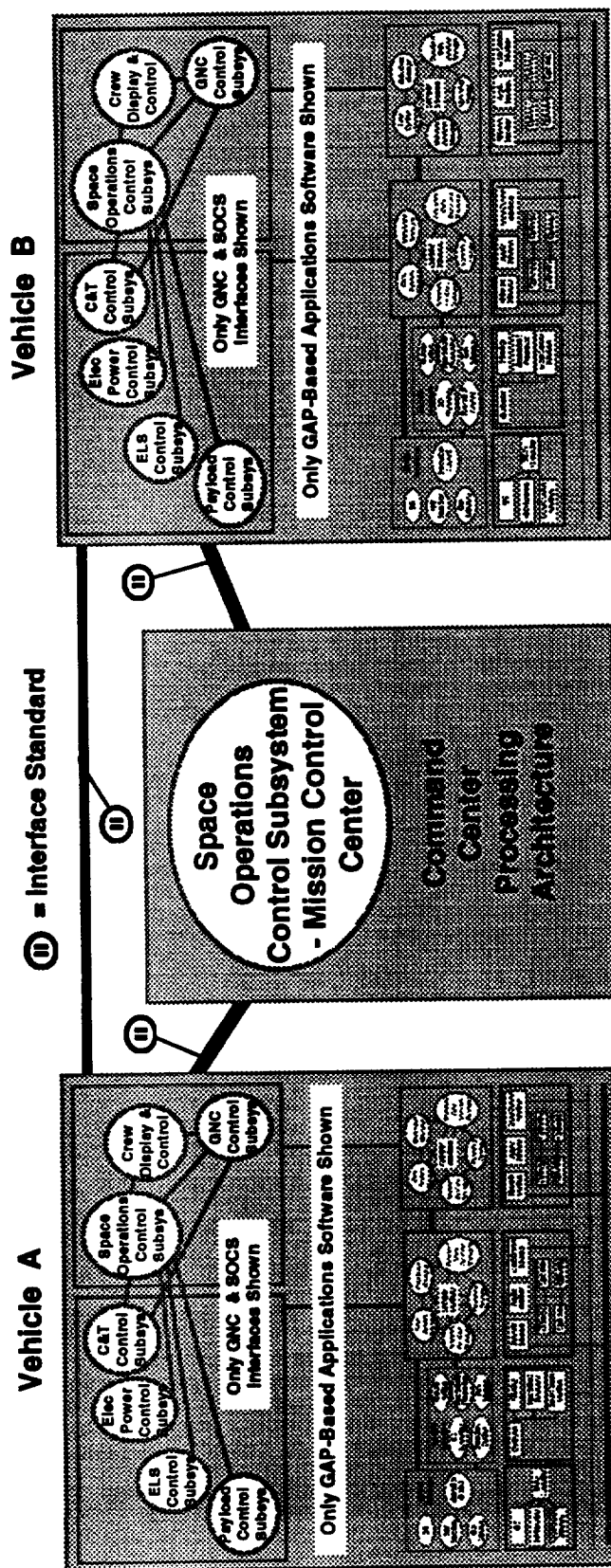


Figure 2-26. Class 6 System A Software-to-System B Software Logical Interfaces



## **2.4 SGOAA RELATIONSHIPS TO POSIX**

As discussed in 2.1.3.1, the top level standard within which the SGOAA was designed to fit is the POSIX OSE Reference Model. The SGOAA extends the POSIX Model beyond the basic objectives of application software portability at the source code (API) level and system interoperability and data portability at the EEI by defining the POSIX interfaces in terms of five SGOAA interface classes (the sixth SGOAA class, Class 2 is an internal interface only), addressing certain Application Platform internal interfaces and recommending additional data system services software specifically applicable to space based systems. User interface look and feel as addressed by POSIX is not presently addressed in the recommended SGOAA standard. The SGOAA is at present primarily oriented toward space based data systems. It is planned that user interfaces be included as a future update.

The POSIX model does not address Application Platform internal interfaces as does the SGOAA. The rationale given in [POSIX91] is that these interfaces have no direct impact on the external interface of a system or the application program interface to the system. System Internal Interfaces are beyond the direct scope of POSIX because they do not directly impact application portability or system interoperability. In addition, there is very little consensus on the partitioning of the platform into components and the consequent allocation of functions to each. In fact, as stated by [POSIX91], this aspect of system design is in a constant and accelerating state of innovation and has been for decades. One of the major objectives of the POSIX API is to decouple the application software from the constantly changing platform. The internal interfaces are not visible to the application software at the API.

Section 2.4.1 provides an overview of POSIX with regard to the SGOAA and section 2.4.2 discusses the relationships of the SGOAA interface classes to POSIX. Figures 2-2 and 2-11 illustrate these relationships. Recommendations are also made with regard to the development of interface standards and to the development of generic open architecture specifications for certain additional Application Platform Language Independent Services.

### **2.4.1 POSIX OVERVIEW**

#### **2.4.1.1 Application Platform**

The Application Platform as shown in Figure 2-11 provides services at the interfaces that , as much as possible, make the implementation specific characteristics of the platform transparent to the application software. All application software entities must access all platform resources via service requests across the Application Platform Interface (API).

Examples of Application Platform elements could include an operating system, a real-time monitor program and all hardware and peripheral drivers. Also included are the SGOAA's Data System Manager, Data Base Manager, Network Services Manager and the Standard Data Services Manager which fall under the category of POSIX Language Independent Services. Application Platform internal interfaces are outside the scope of the POSIX model. The SGOAA defines parts of Class 1(P), 2(P), 3(P) and 4(L) as Application Platform internal interfaces.

#### **2.4.1.2 Application Program Interface (API)**

This is the interface between the application software and the application platform across which all services are provided. It is defined primarily in support of application portability, but system and application software interoperability also are supported via the Communication Services API. The API as defined in POSIX consists of the following parts:

- Communications Services API
- Information Services API
- Human/Computer Interaction Services API
- System Services API

The POSIX Communications Services API, POSIX Information Services API and the POSIX System Services API are required to provide the application software with access to services associated with each of the external environment entities.

##### **2.4.1.2.1 POSIX Communications Services API**

The POSIX Communications Services API is concerned with the interfaces and associated standards that apply to the interface between the application software and the application platform for the provision of communications services to the application software. POSIX is in the process of developing several standards to address this interface. In the SGOAA Standard Data System Services Architecture the provision of communication services is assigned to the Network Manager. This is a SGOAA Class 5 interface.

#### **2.4.1.2.2 POSIX Information Services API and EEI**

The POSIX Information Services API is concerned with both Database Management and Data Interchange Services.

- A. Database Management Services are provided in the SGOAA by the Database Manager. For portability and interoperability, an application using a database must be isolated from the hardware and software retrieval methods as much as possible. The Database Manager provides services to the Application Program via the Data base API. There are currently four database standards, either completed or under development.
- B. POSIX Data Interchange Services provide specialized support for the exchange of data between applications or components of applications. Data interchange standards should define direct formats, data formats, code sets, and data descriptions that are consistent across all implementations of the POSIX Open System Environment to ensure that data can be exchanged between related application software. Data Interchange Services are provided in the SGOAA by the Standard Data Services Manager. These services can be divided into Data Interchange Service API and Data Interchange Services EEI.
  - (1) The Data Interchange API provides an interface from the Application Software to the Application Platform for requesting that specific data be transferred using the EEI services. Little work has been done in developing standards for this interface. No general standards presently exist. This is a SGOAA Class 5(P) interface.
  - (2) The Data Interchange EEI provides an interface from the Application Platform to the EE to support data interchange for storage and retrieval of data using the formats and protocols provided at the Data Interchange EEI.

Standardizing character sets and data representation is crucial to providing effective data interchange between application software operating under differing language and cultural conventions (internationalization).

Standardizing data format protocols protects applications from hardware and/or software differences between environments by ensuring that data remains stable when moving between environments. Data format protocols are fairly well standardized and offer several general standards. This is a SGOAA Class 1(P) interface.

Standardizing data description protocols provides the ability to share data between related applications, even if they were not specifically written to cooperate. To date, most standards in this area have limited themselves to specific application areas and no general solution has been provided. This is a SGOAA Class 4 (L) interface.

#### **2.4.1.2.3 POSIX System Services API**

The POSIX System Services API provides access to the platform internal resources and via the POSIX EEI Communications Services API to the internal resources of other platforms. In order for the platform to protect system integrity and ensure system database consistency, the application software must access all system resources via system service requests. The formal definition of these requests defines the system services portion of the API. The resources provided may be divided into two types of specifications; i.e., Language Service and System Service Specifications. These specifications are defined as follows:

- A. POSIX Programming Language Specifications - Defined by POSIX as the specifications associated with the language such as program control, math functions, string manipulations, etc. A consistent interface to the operating system is essential for applications portability. In addition, the application should be developed using language supported by a standard (preferably international) and system development tools such as a language certified compiler to achieve source code portability. The SGOAA does not address this specification type.
- B. Language-Independent Service Specifications - Consists of POSIX defined services provided to the application software by the underlying application platform internal resources and independent of any programming language. Examples include interprocess communications, interobject messages, access to the user interface, and data storage. Specifications for these services are defined independently of any programming language, and are identified as language-independent service specifications. SGOAA defined services expand this definition and add the Standard Data Services Manager and the Data System Manager. The Standard Data Services Manager is the system services interface from the Applications Platform to the Applications Software for all Application Platform Services. The functions of the Data System Manager are discussed by POSIX under Information System Management and are categorized as an OSE Cross-Category Service. Cross-Category Services are those that may influence and/or impact other parts of the POSIX architectural building blocks.

The three additional SGOAA SDSS Managers (Network, Database, Operating System) defined by the SGOAA are addressed by POSIX and partitioned to other parts of the API. The SGOAA Database Manager is addressed by POSIX Information Interface Services and the SGOAA Network Services Manager by POSIX Communications Interface Services. The Operating System is addressed by POSIX System Services. The purpose of the first in a set of planned POSIX standards, ISO/IEC 09945-1 (IEEE 1003.1) is to define a standard Operating System interface in order to support application portability at the source level.

The Guide to the POSIX Environment [POSIX91] Section 4.2 discusses the emerging standards that are being developed to satisfy the system service requirements. With regard to the SGOAA Interface Model, this API consists of the following two interface classes:

- A. The Operating System to Other code (i.e., Services or Applications) is a SGOAA Class 3 interface.
- B. All other System Service Software to Applications are SGOAA Class 5 interfaces.

#### **2.4.1.3 External Environment**

The EE comprises the external entities with which the application platform exchanges information. These entities are classified into the general categories of human users, information interchange components and communication components.

#### **2.4.1.4 External Environment Interface (EEI)**

The three services present at the EEI are Communications Services, Information Services, and Human/Computer interaction Services.

##### **2.4.1.4.1 POSIX EEI User Interface**

This is the boundary across which physical interaction between the human being and the application platform takes place. Example services across this interface include CRT displays, keyboards, mice, and audio input/output devices. This is a SGOAA Class 1 (P) Interface.

#### **2.4.1.4.2 POSIX EEI Information Interface**

This is the boundary across which external persistent storage service is provided, where the Data Description Protocols and Data Format Protocols are required to be specified for data portability and interoperability. This is a SGOAA Class 1 (P) Interface where Data Format Protocols are a hardware function. Data Description and Data Format Protocols (software) place this interface into Class 4 (L) for System Software-to-System Software communications. Data Description Protocols place this interface into Class 6 (L) for Application-to-Application communications.

#### **2.4.1.4.3 POSIX EEI Communications Interface Services**

This is the interface that provides access to services for interaction between internal application platform software entities and application platform external entities such as application software entities on other application platforms, external data transport facilities, and devices. The services provided are those where protocols and formats must be standardized for interoperability.

This hardware-to-hardware direct interface is a POSIX EEI interface and a SGOAA Class 1 (P) interface. The standards at the EEI will be in several areas such as physical connections, network protocols/formats and distributed system services. Much standardization work has gone into the aspects of networking that are available at the EEI. The standards selected at the EEI will impact system interoperability, but may also have an effect on application portability, because certain applications may require particular types of network access to function.

The interface from the software driver to the communication hardware (Hardware-to-System Software Direct) is an Application Platform internal interface defined as an SGOAA Class 2 (P) interface.



Table 2-9. SGOAA Relationships to POSIX

POSIX INTERFACES	SGOAA CLASS					
	1 (D)	2 (D)	3 (D)	4 (L)	5 (D)	6 (L)
<u>API:</u>						
System Services			X		X	
Communications Services					X	
Information Services					X	X
User Services					X	
<u>EEL:</u>						
Communications Services	X					
Information Services	X			X		X
User Services	X					
<u>AP Internal:</u>						
Backplane Bus	X					
System SW to System SW			X	X		
HW to System SW		X				

## 2.4.2 SGOAA INTERFACE CLASS RELATIONSHIPS TO POSIX

The POSIX OSE Reference Model is the top level standard within which the SGOAA must fit. Table 2-9 shows the SGOAA Interface Class relationships to the POSIX Model interface definitions. The following paragraphs discuss the rational for establishment of these relationships.

### 2.4.2.1 Class 1 Hardware-to-Hardware Interfaces (DIRECT)

SGOAA interface definitions and standards requirements identification activity are not recommended to be limited to only those interfaces defined by the POSIX model. Hardware modularity and portability in addition to the application software portability and interoperability addressed by POSIX should also be considered. In order to provide programs access to multiple sources of hardware components and to enable maintainability and technology upgrades of application platforms over extended life cycles, especially in the case of space avionics, it is recommended that the Application Platform internal hardware interface standards for the SGOAA Class 1 "Backplane Bus" interfaces shown in Figure 2-16 be imposed. These standards should not be imposed to place unnecessary constraints on the platform design, rather they should be imposed to require vendors to select a bus that has

platform design, rather they should be imposed to require vendors to select a bus that has wide industry support. The Network Interface, Local Communication Interface, and the Sensor/Effector interface shown in Figure 2-12 are all SGOAA Class 1 interfaces and fall within the definition of POSIX EEI Communications Services interfaces. It is recommended that standards be established for the following SGOAA Class 1 interfaces:

- Backplane Bus (Application Platform Internal Interface)
- Network Interface (POSIX Communications EEI)
- Local Communication Interface (POSIX Communications EEI)
- Special I/O Processing - Sensor/Effector (POSIX Communications EEI)

It is not recommended that an interface standard be applied to the Instruction Set Architecture (ISA) as shown in Figure 2-16 (Application Processing Logical Standards). The technology is changing too rapidly and imposing such a standard would place undue restrictions on the design of the application platform. Compliance to POSIX and an approved "Backplane Bus" standard will enable selection of the best ISA Standard implementation to satisfy system requirements and also achieve the hardware maintainability, upgradeability and portability objectives.

POSIX EEI Information interface Data Format Protocols are a Class 1 interface where implemented in hardware.

The POSIX EEI User Interface is the boundary across which direct interaction between the human and Application Platform takes place.

#### **2.4.2.2 Class 2 Hardware-to-System Software Interface (DIRECT)**

This interface is that of the Operating System (OS) binding of the hardware driver software to invoke platform services as shown in Figure 2-18. It is an interface internal to the Application Platform. The Standards for this interface can be separated into the OS Standards and the Hardware Interface Standards for a specific hardware implementation. The hardware interface standards referenced in Class 1 must be definitive as to the software driver interface requirements needed to communicate with that hardware. Each OS Specification/Standard must specify the software interface binding requirements for "plugging" a driver into the OS. POSIX defines this interface as a "Layering" or Redirection" service. It is recommended that a interface standard be developed for the OS binding to

software driver side of this interface. Defining a standard for this interface is not presently a part of the POSIX standardization activity.

#### **2.4.2.3 Class 3 System Software-to-Software (Local DIRECT)**

This interface is included in the POSIX API, the interface between the Application Platform and the Application Software against which all platform services are requested and provided. It is defined primarily in support of Application Software portability.

The System Services API provides access to services associated with the Application Platform internal resources. The System Services API is classified as a SGOAA Class 3 interface as illustrated in Figure 2-20 as the Application Software has a direct interface to Operating System Services. This distinction is made since the SGOAA architecture separates Operating System Services (SGOAA Class 3) from other System Services (SGOAA Class 5). SGOAA Class 3 is also classified as a POSIX Application Platform Internal Interface due to System Services Software having a direct interface to Operating System Services.

#### **2.4.2.4 Class 4 System Software-to-System Software Interfaces (LOGICAL)**

This is the internal interface for transfer of data between Application Platform Language Independent System Services as illustrated in Figure 2-22. It is recommended that an interface standard specifying data description protocols be developed to provide Language Independent System Services data portability and interoperability. This is also the external logical interface between Language Independent System Services on the Application Platform with Language Independent System Services on other Application Platforms. For example, this is the logical interface between the Data Base Manager in a Application Platform communicating with the Data Base Manager in another platform. An example for the Space Station Data Management System (DMS) is a user requesting a file transfer by way of the RODB in a Standard Data Processor (SDP) which requests the service to be performed by the file transfer server in the Mass Storage Unit (MSU).

POSIX EEI Information/Data Interchange Services data description protocols are defined as SGOAA Class 4 for data interchanges between System Services Software both within and between Application Platforms. These services provide the ability to share data between related System Software entities, even if they were not meant to specifically cooperate.

Building upon the Data Format Protocols and Character Sets and Data Representation Protocols defined as Class 1, these data description protocols provide a means of associating a name or other identifier with the individual data elements in a standard manner. Most standards developed in this area have limited themselves to specific application areas with no general solution provided.

#### **2.4.2.5 System Software-to-Application Software Class 5 (DIRECT)**

The SGOAA Class 5 Interface is shown in Figure 2-24. This interface is the POSIX API, the interface between the Application Platform and the Application Software against which all platform services are requested and provided. It is defined primarily in support of Application Software portability. System and Application software interoperability are also supported by the Communications Services API. The following four APIs are in this POSIX interface:

- System Services API
- Communications Services API
- Information Services API
- Human/Computer Interaction Services API

All of the preceding API's are SGOAA Class 5 interfaces. The System Services API provides access to services associated with the Application Platform internal resources and the other three APIs provide the Application software with access to services associated with each of the EE entities. The System Services API is also classified as a SGOAA Class 3 interface as the Application Software has direct interface to Operating System Services. This distinction is made since the SGOAA architecture separates Operating System Services from other System Services.

#### **2.4.2.6 Class 6 Application Software-to-Application Software (LOGICAL)**

##### **2.4.2.6.1 Local or Node Applications Software Interfaces**

Figure 2-25 illustrates SGOAA Class 6. This is the internal interface for transfer of data between Application Software within an Application Platform and as such is a POSIX Information Services API interface. This is also the external logical interface between Application Software on the Application Platform with Application Software on other Application Platforms and as such is a POSIX Information Services EEI interface. It is

recommended that interface standards specifying data description protocols be developed to provide Application Software data portability and interoperability.

POSIX EEI Information/Data Interchange Services data description protocols are defined as SGOAA Class 6 respectively for data interchanges between Application Software both within and between Application Platforms in the same system and in external systems. These services provide the ability to share data between related Application Software entities, even if they were not meant to specifically cooperate. Building upon the Data Format Protocols and Character Sets and Data Representation Protocols defined as Class 1, these data description protocols provide a means of associating a name or other identifier with the individual data elements in a standard manner. Most standards developed in this area have limited themselves to specific application areas with no general solution provided.

#### **2.4.2.6.2 System-to-System Applications Software Interfaces**

As shown in Figure 2-26, this is the external logical interface between Application Software on an Application Platform in a system with Application Software on other Application Platforms in external systems and is a POSIX Information Services EEI interface. It is recommended that interface standards specifying data description protocols be developed to provide Application Software data portability and interoperability.

POSIX EEI Information/Data Interchange Services data description protocols are defined for data interchanges between Application Software running on a Application Platform in one system with Application Software running on Application Platforms in one or more external systems. These services provide the ability to share data between related Application Software entities, even if they were not meant to specifically cooperate. Building upon the Data Format Protocols and Character Sets and Data Representation Protocols defined as Class 1, these data description protocols provide a means of associating a name or other identifier with the individual data elements in a standard manner. Most standards developed in this area have limited themselves to specific application areas with no general solution provided.



### **3. THE SPACE GENERIC OPEN AVIONICS ARCHITECTURE APPLIED**

Development of the architecture reference system model, hardware model and interface class model were discussed in Section 2. Development of lower level detailed SGOAA functional architectures is discussed in this section. Development of the SDSS Subsystem and SOCS functional architectures were parallel tasks. Each nourished and provided reality to the other. The following ground rules were established for the functional architecture developments:

- The architecture analysis will be based on mission needs in an integrated approach.
- The architecture must be an open generic architecture that can be applied to multiple space missions and programs.
- An open architecture should be applied down to the module level where possible.
- The architecture must be a modular architecture in which the elements are autonomous, coherent and organized in a robust structure. Robustness is the ability of systems to continue functioning under abnormal conditions.
- The architecture must provide systems with the extensibility to be extended or adapted to new conditions, changes in specifications or new technology.
- Differentiation between processing levels will be based on the philosophy of "Centralized Command and Decentralized Execution".
- The architecture must be a "shopping list" of all processes applicable to any space vehicle or other planet base.
- This architecture will be compatible with the avionics software POSIX OSE Reference Model.

With regard to the OSE Reference Model [PRU90] and [POSIX91], there are three types of entities used in the OSE model: Application Software, Application Platform and External Environment. The four types of interfaces defined in the OSE are user, information exchange, communications, and processing. Definitions of the OSE entities and interfaces are discussed in Appendix A of this document. The six classes of SGOAA interfaces described in Section 2.3.2 of this document titled "Architecture Interface Model Description" are referred to throughout this section.

This section discusses the development of the Space Generic Avionics (SGA) Core Functional Architecture using the model described in Section 2.3, describes the detailed subsystem functional architectures (i.e., subarchitectures) developed and provides functions and data flows for two of the key integrating subsystems, the SDSS and the SOCS. It also provides application examples where the SGOAA could be applied to real world space avionics.

### **3.1 POTENTIAL SPACE GENERIC AVIONICS FUNCTIONS**

The first step in defining space generic avionics functions was to gather together space platform related processes, events, major data items, and any other information which can be used to identify logical groupings (i.e., entities) of information handling. These groups of information define user needs for processes and the related data being processed. Multiple space program experience of the architecture study participants was used to research Space Station Freedom and Shuttle sources and identify/document the required data. The results of this research activity is shown in Figure 3-1, a table of space system user processes grouped together by subsystem function. These processes are not intended to be a definitive list of all space processes, but are a functional checklist of processes against which the subsequent requirements, analyses and the resulting generic avionics architecture can be compared to determine if all "traditional" space functions are accounted for or accommodated.

This functional checklist also suggests some of the partitioning into higher level entities. A vehicle control entity appears needed to coordinate the subsystem applications operating on the vehicle and to supervise their activities. This vehicle control system would also provide a means of human operator coordinated control over all vehicle applications or operations. It also suggests that another higher level function needed is one of operations control to coordinate all activities and processing inside the space vehicle with each other and with outside activities and processes. The operations control layer would also provide the place for logic "glue" as needed to enable the activities and processes to respond to external intervention from the crew or remote commanding.



The shadowed elements in Figure 3-1 represent a shopping list of functions required in present space vehicles (i.e., Space Station Freedom or Space Shuttle) data management or processing systems. Note that this structure does not correspond exactly to the station software partitioning, and is not precisely the same as the implied partitioning in [JSC 31000]. The purpose of the structure is to focus attention on data service requirements and related or ancillary processing requirements.

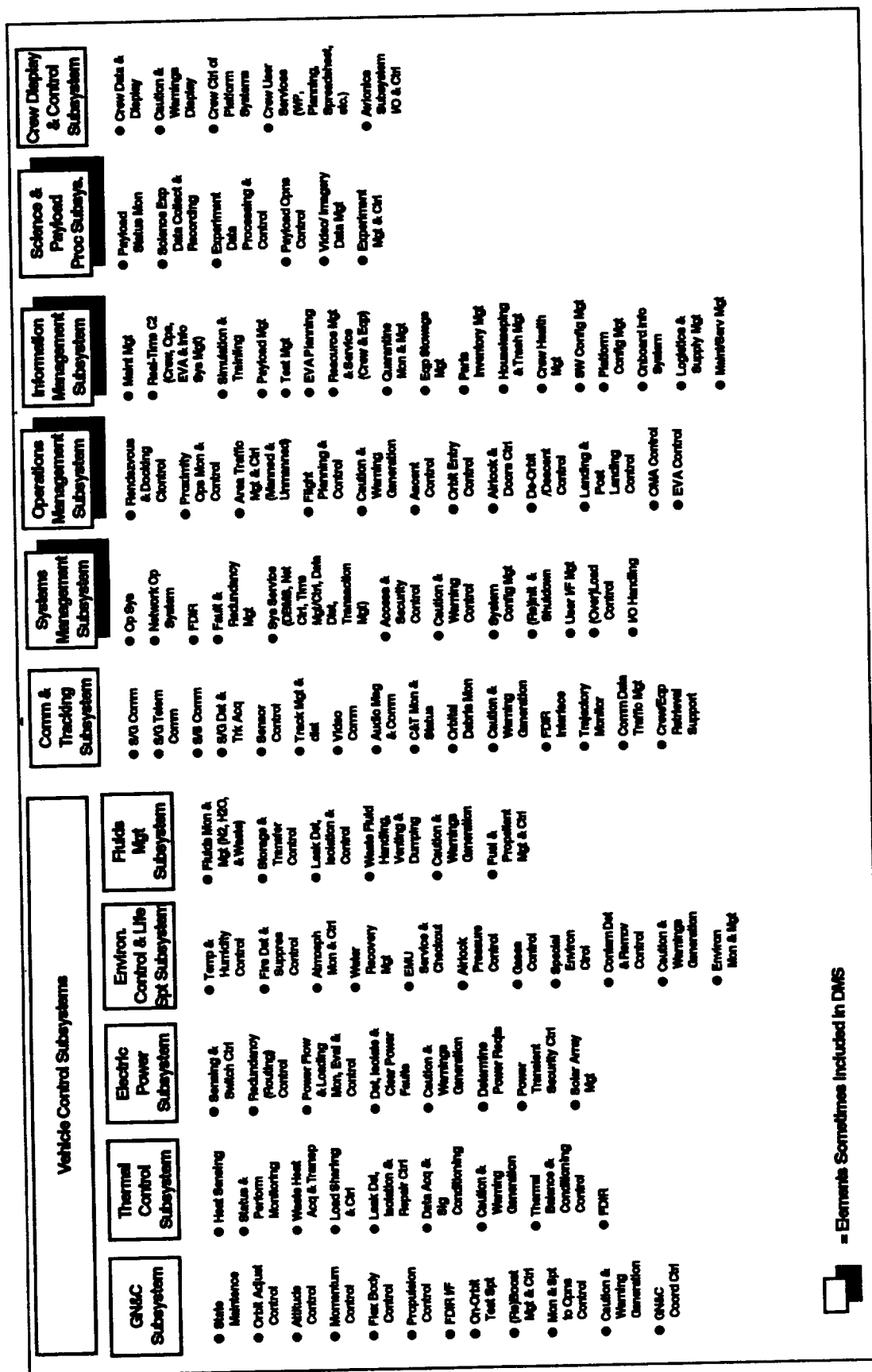


Figure 3-1. Space Generic Avionics Potential Functions Checklist

### **3.2 DATA AND CONTROL FLOW DIAGRAM CONVENTIONS**

The architecture that is described in the following paragraphs is represented by merged data and control flow entity diagrams. These diagrams are often referred to as process flow diagrams, which are somewhat different in fact. This architectural discussion will use bubbles to illustrate and describe the process entities. In the "Hybrid Object Oriented Structure Analysis" process used to develop this architecture and described in reference [WRA91], abstracted processes and data are referred to as entities to distinguish them from the object oriented analysis (OOA) objects. The process entity bubbles may represent either data processes or control processes. Although referred to as data/control flow entity diagrams, the bubbles are thought of as entities (with noun names) to clarify that they encompass more than just data or control processes, and include other requirements more related to object oriented development as described below:

- Definition of objects is by abstracting the processes and data, and establishing the services which operate on the data based on inputs to the object.
- Data attributes are defined for each object and similarly for each hybrid approach entity.
- Services are the processes performed as a result of messages received by the object. In the hybrid approach, services are more system process oriented.
- OOA suggests the use of assemblies which are component parts of an object broken down into lower level objects; this is similar to the hybrid approach's structured breakdown of entities into entities at lower levels or sub-entities.
- The hybrid approach allows only one class membership, namely that higher level entity which spawns the sub-entity.
- Information hiding is achieved in the hybrid approach by defining external interfaces and services for each entity which are the only access points for that entity.
- In the Hybrid Object Oriented Structural Analysis approach, inheritance is achieved by defining that the requirements for an entity automatically apply to all lower level entities. If a requirement does not automatically extend to all lower level entities, then it would not show up at the higher level, but would only be attached to the lower level entities to which it applied.



### 3.3 FUNCTIONAL ARCHITECTURES

Figure 3-2 defines the boundaries of the SGA Core Functional Architecture. The SGA Core Functional Architecture consists of the avionics systems as a black box surrounded by external elements with which it interacts. The black box consists of all hardware, software and other electronics needed to control and operate the space vehicle, and provides the coordinated functionality for end-to-end processing in handling the information needed to use the black box's elements, to control its interaction with its environment and to respond to human commands. The SGA black box provides the capability to meet the top level user requirements.

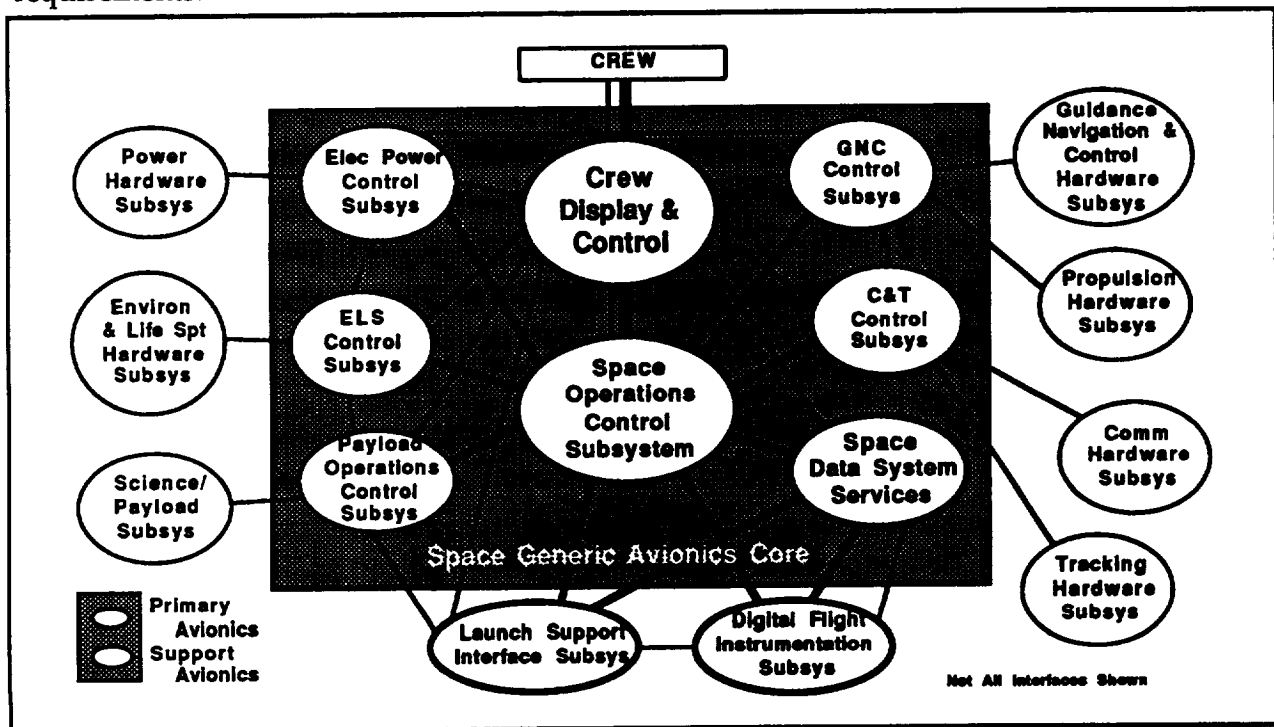


Figure 3-2. Space Generic Avionics Core Functional Architecture

Within the black box are the primary functional entities which enable the avionics to support the mission and sustain the crew. These primary functional entities include the traditional applications control subsystem entities consisting of Electric Power; Environment and Life Support; Payload Operations; Guidance, Navigation and Control; and the Communications and Tracking Control Subsystems. The typical support avionics are shown with bold outlines, some of which are core avionics and some of which serve specialized non-core functions. The core avionics functions that are also bolded in the figure are the SOCS application (which integrates all activities from the traditional applications control subsystems to serve the crew), the SDSS subsystem (which provides

applications control subsystems to serve the crew), the SDSS subsystem (which provides data processing and data communications support to all the traditional and operations control subsystems), and the Display and Control subsystem (which enables the crew to interface with and direct the avionics). Development of the functional architectures for the SOCS and the SDSS are the primary focus of this section. Development of a functional architecture for the crew's Display and Control subsystem is a recommended future task.

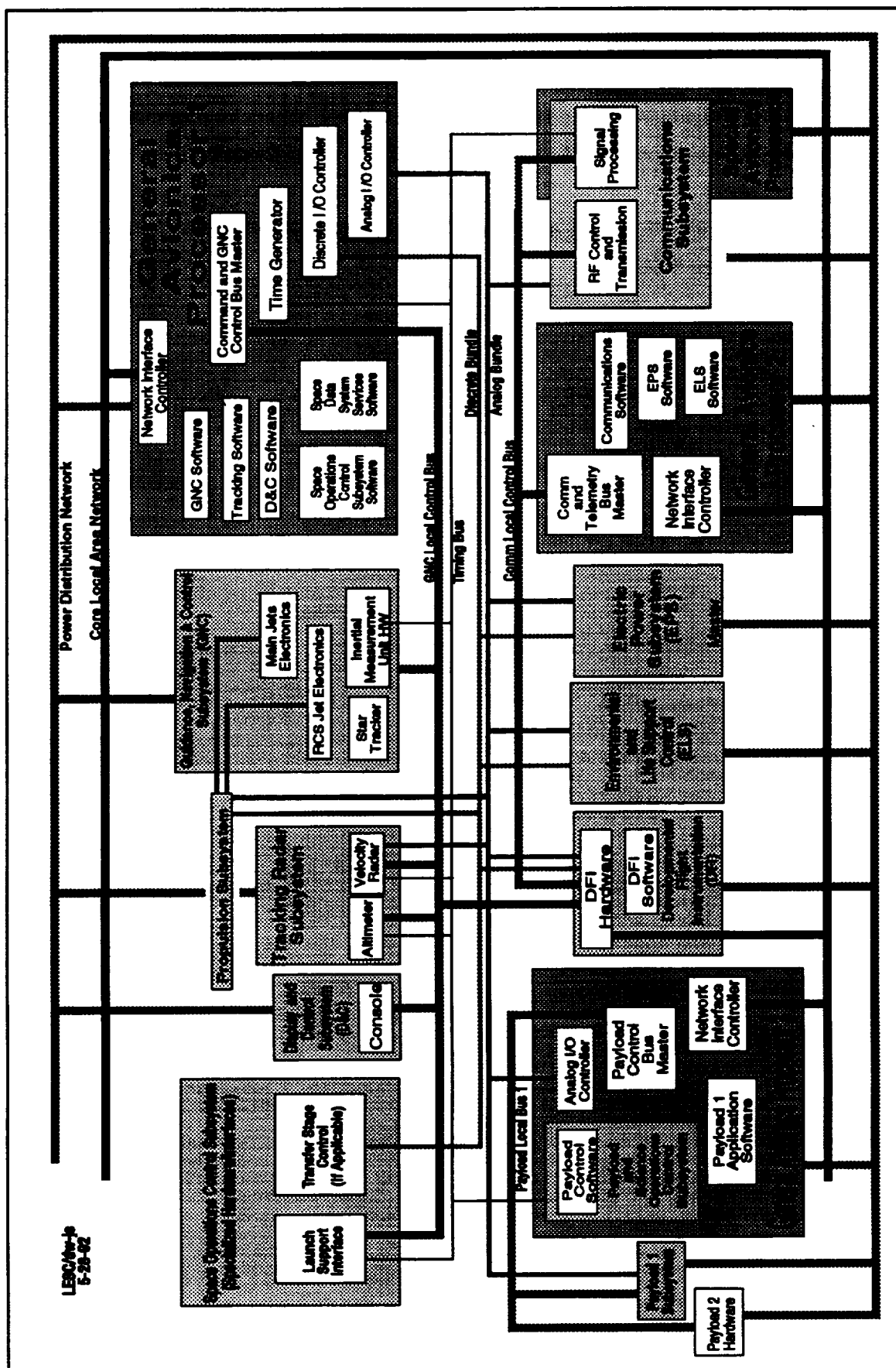
The system architecture for the hardware and software components may be merged with these functional architecture elements to produce the composite architecture diagram shown in Figure 3-3.

Note that it is not possible to develop such a diagram for the completely generic case, because some assumptions must be made on decisions such as how many processors are needed and how functionality is to be allocated among the processors. Around the outside of the elements in this figure are the power distribution and core local area networks which tie together the separate subsystem units. In the center of the elements are the multiple local control buses which tie together the elements within a subsystem unit. This figure assumes that three general avionics processors, one special avionics processor, three local control buses, a timing bus, and one set each of analog and discrete bundles are needed. Two stand-alone payloads are shown under the control of the payload subsystem controller. Not all hardware and software elements are shown; just enough to show how the generic architecture could be allocated and tailored to a real, operational mission and system.

Specialized SOCS hardware elements for operational command and control of the vehicle and vehicle launch checkout and control are also shown, such as the launch support interface (and perhaps launch blockhouse elements depending on functional allocations), and the controller for the transfer stage (if it is controlled from the spacecraft and not independently from the ground). GAP 1 is assumed to be adequately sized to support the integrated processing needed for guidance navigation and control, tracking, display and control, command processing in SOCS, and the data system in SDSS. It includes the appropriate input/output for analog and discrete signals for the entire system, since (it is assumed) that commands over the network to other subsystem units would be complemented by analog or discrete control signals over these lines to set up the other subsystem units. GAP 2 is assumed to operate in conjunction with the remaining traditional subsystems (communications, electric power, and environment and life support; as well as with the developmental flight instrumentation needed on early missions. GAP 2

operates (in this assumed allocation) in conjunction with a special avionics processor used to support communications or tracking signal processing. GAP 3 supports the operation of payloads, with some applications running in this GAP and others perhaps running in specialized payload subsystems such as represented by Payload System 1.

The next section describes the SDSS and the SOCS, the two key subsystem architectures developed to define the SGOAA.





### **3.4 KEY INTEGRATING SOFTWARE SUBSYSTEM ARCHITECTURES**

The scope of this architecture study to date has been focused on space data system architectures. To develop space data system architectures for the NASA JSC Flight Data Systems Division, it was necessary to examine the key applications which drive space data system architectures, i.e., the space operations applications. These key applications consist of the Crew Display and Control Subsystem and SOCS. These subsystems, as identified in Figure 3-2, are the two operationally oriented applications falling within the auspices of the NASA JSC Flight Data Systems Division needed to meet user needs and which also provide the primary means by which astronauts control the operation of the spacecraft. They must be effectively supported by the SDSS, also within the auspices of the NASA JSC Flight Data Systems Division. The SDSS consists of all data processing and data communications services and operating system entities supporting the vehicle avionics subsystems. The SOCS and SDSS Subsystem architectures have been developed and are presented in the following paragraphs of this section. The Crew Display and Control Architecture is recommended as a future development task for the SGOAA study. The other avionics subsystems identified in Figure 3-2 fall under other NASA JSC Divisions. It is recommended that the SATWG initiate development of generic avionics architectures for those avionics subsystems identified in Figure 3-2 that are outside the auspices of the NASA JSC Flight Data Systems Division.

The SDSS and the SOCS are the two primary integrating subsystems in a space vehicle avionics architecture. The following paragraphs discuss the generic architectures developed for these two subsystems and present the top level merged data and control flow diagrams. Common elements that provide a broad spectrum of generic avionics functions/services are also presented. To be developed as complete architectures for specific applications, the generic architecture definitions for these two subsystems require additional system engineering activity to develop control state transition diagrams, fault tolerance/redundancy management requirements and risk management definitions, perform prototyping and simulation implementation, and implement performance analysis.

#### **3.4.1 SPACE DATA SYSTEM SERVICES ARCHITECTURE**

The SDSS is a generic architecture designed to provide a comprehensive set of data processing services to all space vehicles and subsystems. The SDSS architecture was designed to satisfy the following architectural guidelines:

- Services must hide hardware implementation characteristics from applications.
  - Minimize change effects and maintenance problems.
  - Ease reconfiguration.
  - Reuse software.
- Use standards for services to achieve open architecture.
  - Provide transparent system expansion.
  - Enable new technology insertion.
  - Minimize interface implementation difficulties and change.
  - Provide a basis for interoperability.
- Based on Client Server Model
  - Application Program Interface (API) must be established and utilized to achieve application portability.
  - Message protocol must be standardized to achieve interoperability between cooperating systems.
- Hides Hardware Characteristics from Applications
  - Minimize change effects.
  - Ease of reconfiguration.
  - Software reusability.
  - Minimize software maintenance.
- Hide System Configuration from Applications
  - Minimize change effects.
  - System expansion transparency
  - New technology insertion transparency.
  - Minimize software maintenance.
- Use Standards Where Feasible
  - System expansion transparency.
  - New technology insertion transparency.
  - Minimize interface changes.
  - Allows utilization of common items.
- Avoid Duplication of Function
  - Minimize training.
  - Standardizes interfaces.
  - Minimizes development costs.
- Provide Common Service Elements
  - Where needed by two or more applications.
  - Common software.
  - Generalized services to connect all systems.
- Transparency of Data Location and Source
  - All access is through Runtime Object Data Bases (RODBs) located in distributed processing elements.
- Supports a Multi-Processing/Distributed Processing Environment.
- Fault Detection and Redundancy Management
  - Built in to all levels of the architecture.
  - Hardware monitor of hardware failures.
  - Software monitor of system status.
  - Automatic reconfiguration by hardware/software with manual override.

Figure 3-4 shows the five functional entities which comprise the SDSS to meet these architectural requirements.

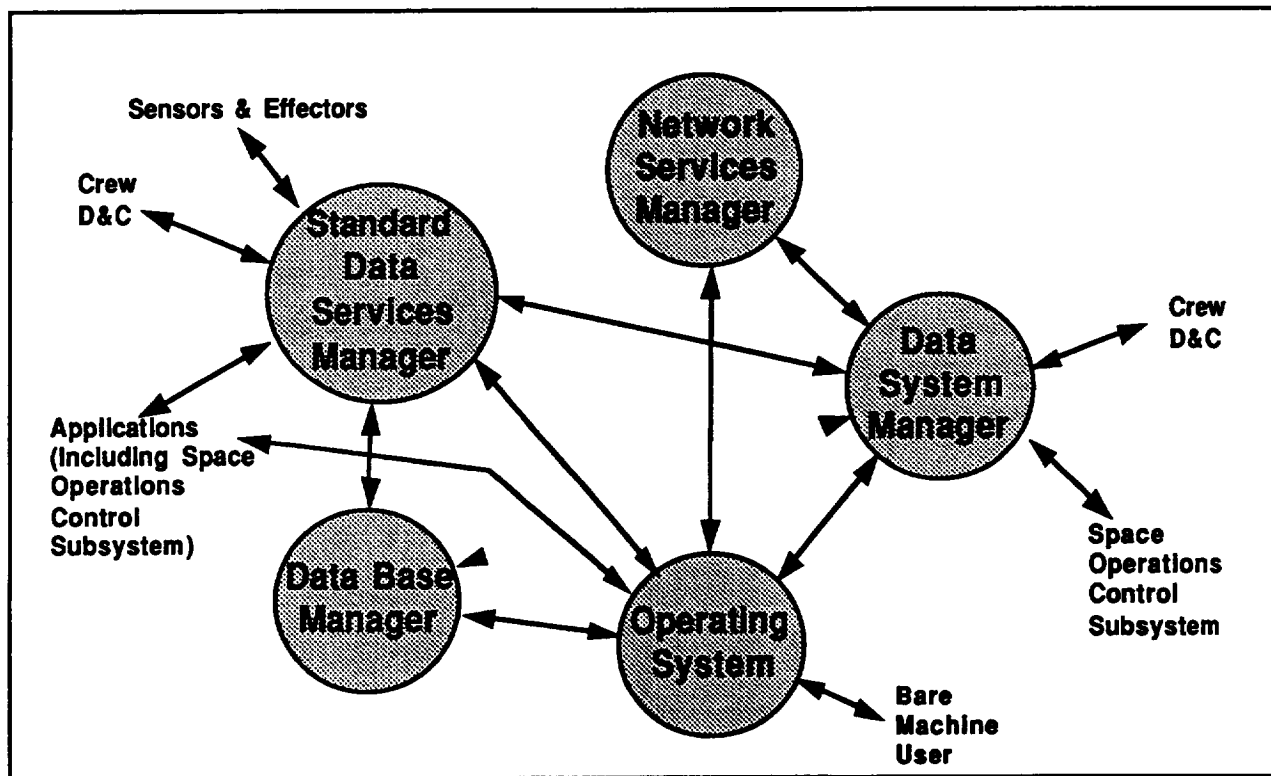


Figure 3-4. Space Data System Services

The **Standard Data Services Manager** provides all interface to the system users for data processing and data communication services. Services to be provided to the users are derived directly from user requirements.

The **Data System Manager** provides the housekeeping and control services for the SDSS. There is a command and control interface to the crew and to the SOCS. Command and control service requirements are derived directly from user needs.

The three other software entities in the architecture, **Network Services Manager**, **Data Base Manager** and **Operating System** provide SDSS services as required by the Standard Data Services Manager and the Data System Manager. An Ada Language bare machine user may interface directly with the Ada RTE.

Note that Figure 3-4 identifies both direct and indirect derived required functions. Direct derived required functions (Standard Data Services Manager and Data System Manager) are

those derived to directly provide the functionality needed to meet requirements imposed by user applications for information and services control and data distribution. The indirect derived required functions (Network Services Manager, Data Base Manager and Operating System) are derived, in turn, to meet the requirements imposed by the Standard Data Services Manager and Data System Manager functions; thus they indirectly meet user applications requirements for control.

The successful implementation of a generic avionics architecture is based on the establishment and compliance to both logical and physical interface standards and requirements at the user interfaces to that architecture and internal to the architecture. The following paragraphs discuss each of the five SDSS functional entities at lower levels of functional decomposition.

#### 3.4.1.1 SDSS Control Modes

Figure 3-5 shows the multiple ways in which a user may exercise control of and communicate with the SDSS. These multiple logical communication paths provide robustness to the system.

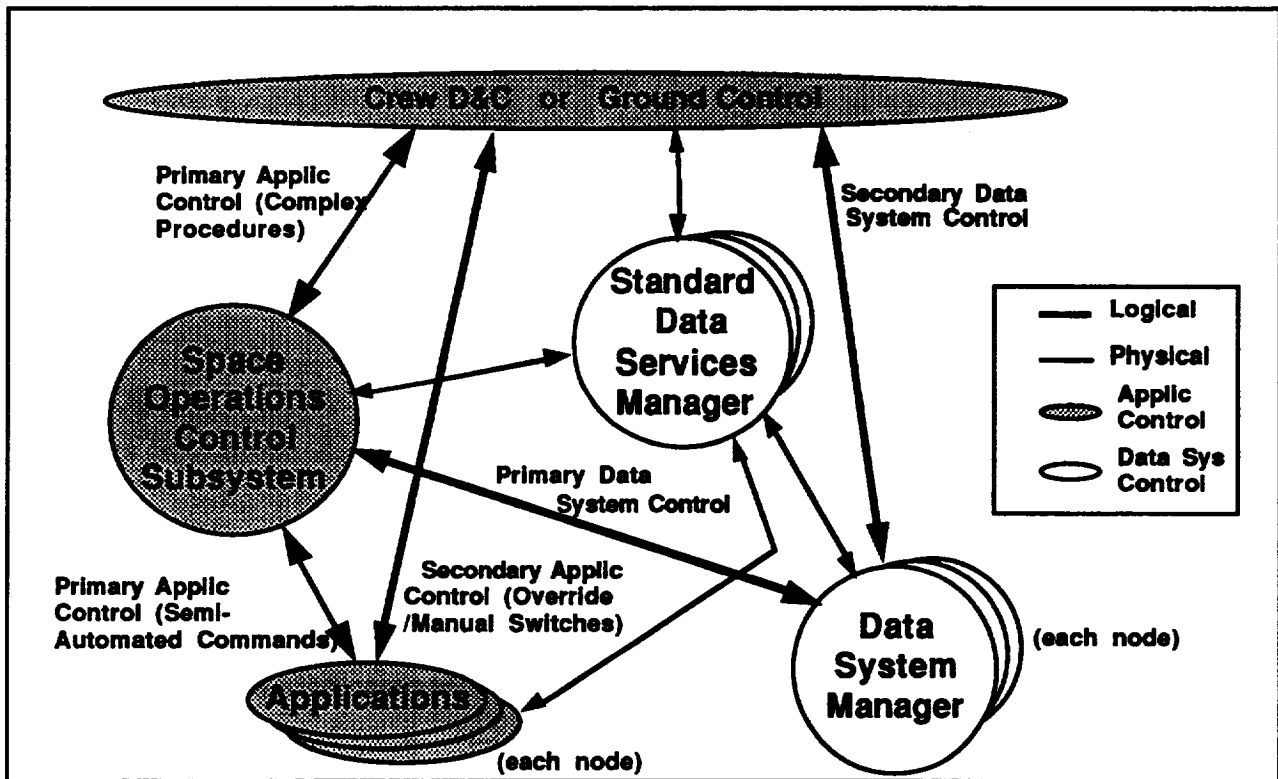


Figure 3-5. Space Data System Services Provides Multiple Control Modes

There are direct hardware (physical) interfaces established between the Crew Displays and Controls (D&C), Ground Control, SOCS, Applications and the Standard Data services Manager. Applications are defined as user software running in a processing element and any associated hardware under control of the application such as Inertial Measurement Units, RCS jets, etc. The ground control interfaces are by way of RF links to a SAP and then over the core network or local communication to the processing element containing the Standard Data Services Manager of interest. Crew, SOCS and Application interface is over the core network, local bus or by a direct interface to the processing element.

User transparent logical interfaces provide alternate paths for command and control of the SDSS and Applications. The primary logical control path is through the SOCS to either the Applications or to the Data System Manager. In the event of a SOCS malfunction, the crew can communicate directly with the Data System Manager or Applications by way of service paths provided by the Standard Data Services Manager. Manual overrides such as switches may be provided as required.

Note that the single physical interface to the external environment is the Standard Data Services Manager thus simplifying the physical interface verification process.

#### **3.4.1.2 Standard Data Services Manager**

The Standard Data Services Manager is a service software functional entity that provides data and command service common functions that are required by one or more applications. Data is defined as sensor data, application data and crew or SOCS data. Commands refer to effector commands, requests to application programs and requests to Crew D&C procedures.

As shown in Figure 3-6, Standard Data Services consists of the modular sub-elements Data Acquisition, Data Distribution and the Reports Generator. Standard Data Services executes under the Operating System.

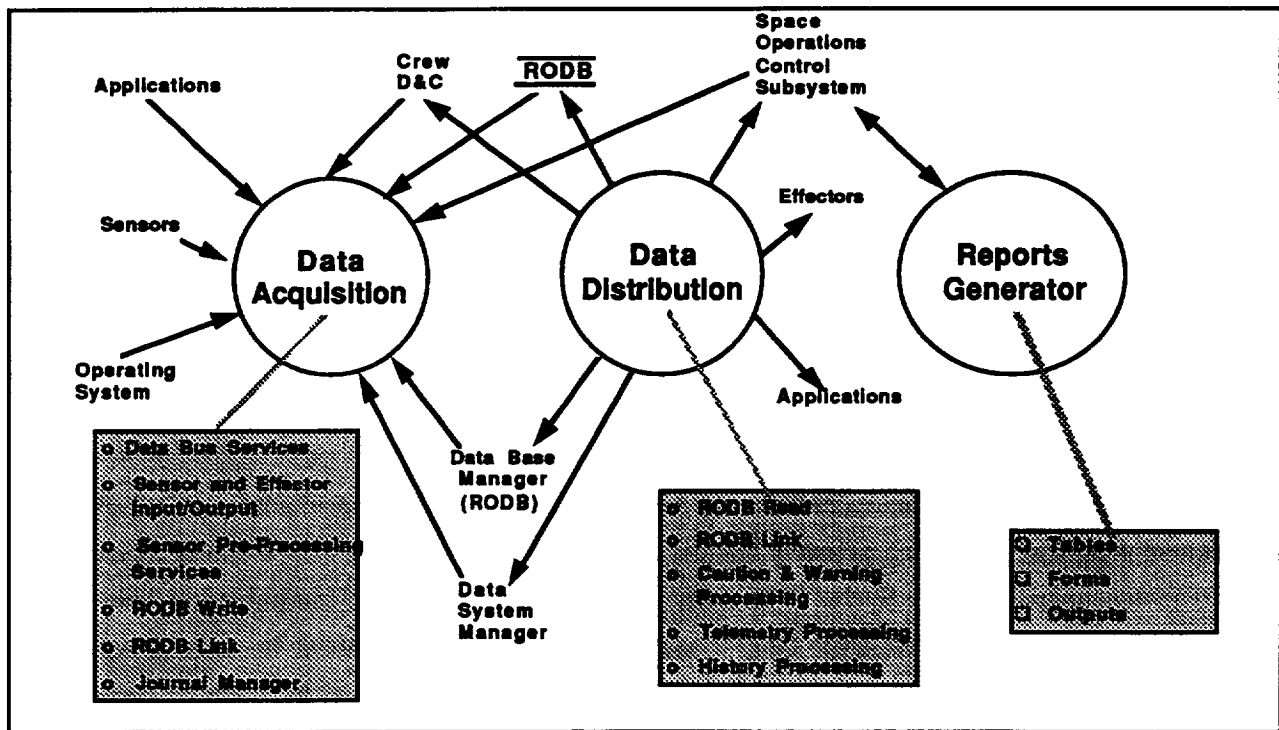


Figure 3-6. Standard Data Services Manager

#### 3.4.1.2.1 Data Acquisition

Data Acquisition (DA) provides both data and command functions. Provided are the necessary protocols and command sequences to acquire data over local data buses. DA places all sensor data in a RODB for application access. DA, at user request, can also perform pre-processing on sensor data such as limit checking and data conversions. DA performs all writes of data and commands to both local RODBs and remote RODBs (RODB Link) as required by applications. All non-recoverable changes to the RODB are sent to the Journal Manager in a distributed processing system in order to maintain a RODB backup copy for recovery purposes. DA provides the necessary protocols and command sequences to write effector commands issued by applications over local data busses. DA also provides the interface to the Network Services Manager for remote effector commanding over the core network to remote processing elements. DA, at user request, provides command data conversions from computed data types to data types required by the effector device. For a distributed processing system, the Journal Manager loads the RODB in the distributed processing elements at initialization and maintains a backup copy of the RODB for reconfiguration and/or reinitialization.

#### **3.4.1.2.2 Data Distribution**

Data Distribution (DD) provides local and remote reads of Standard Data Services data and commands from local and remote RODBs. RODB read supports application reads of sensor data, command requests, and application derived data from both local RODBs and remote RODBs (RODB Link). This function also handles the distribution and logging of all data associated with caution and warning events, application derived events and advisory messages. These events may be detected from sensor data or detected by applications. Events messages are created and distributed. Standard Data Services health and status data is provided to the Data System Manager for evaluation and appropriate action. DD sends RODB sensor data and application derived data to the ground in accordance with specified telemetry data tables and formats.

#### **3.4.1.2.3 Reports Generator**

The Reports Generator provides a general report generation capability primarily based on predefined table driven formats, data sources and report destinations. Requests for reports are made through the SOCS. This entity also provides the capability to generate special purpose reports.

#### **3.4.1.3 Network Services Manager**

The Network Services Manager (NSM) as shown in Figure 3-7 provides for peer-to-peer communication between applications on distributed processing elements communicating over the SDSS core network. These processing elements may be on the same core network, on different core networks (by way of a gateway or bridge) or to a remote system such as the ground through a SAP configured as the service provider for an air to ground communication link. The NSM provides all management services required for the local core network.

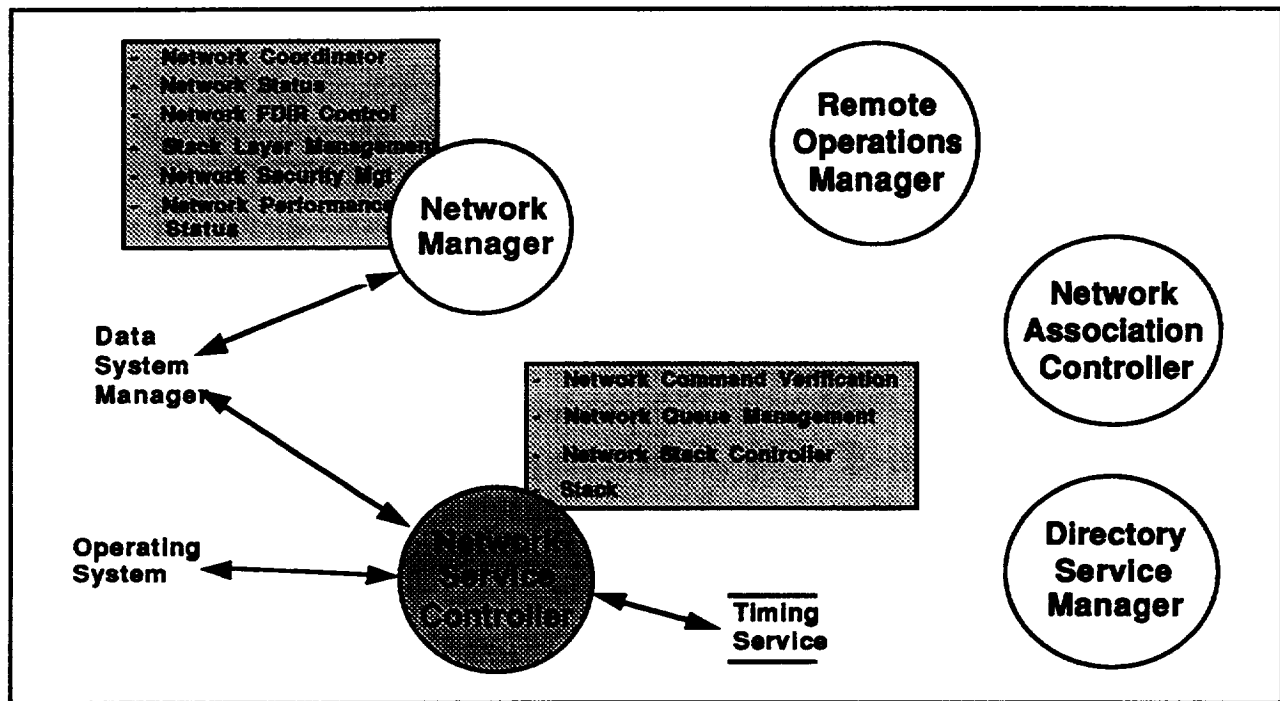


Figure 3-7. Network Services Manager.

The architecture is based on standards developed by, and others currently under development by the ISO. The basic reference model for OSI consists of seven layers. Figure 3-8 shows an example of partitioning between software and hardware supported by this architecture. The NSM architecture is targeted toward having a common implementation executing under the Operating System in each processing element on the core network.

As shown in the example of Figure 3-8, a potential relationship can be mapped between the OSI Stack and elements of the NSM architecture. The OSI stack implementation is a design that can be implemented under the NSM architecture; however, the NSM architecture is general enough to allow other implementations such as TCP/IP. This is accomplished by providing for capability for the NSM architecture to contain software interfacing up to the applications, software interfacing between applications and drivers, software drivers for the hardware, and the actual hardware.



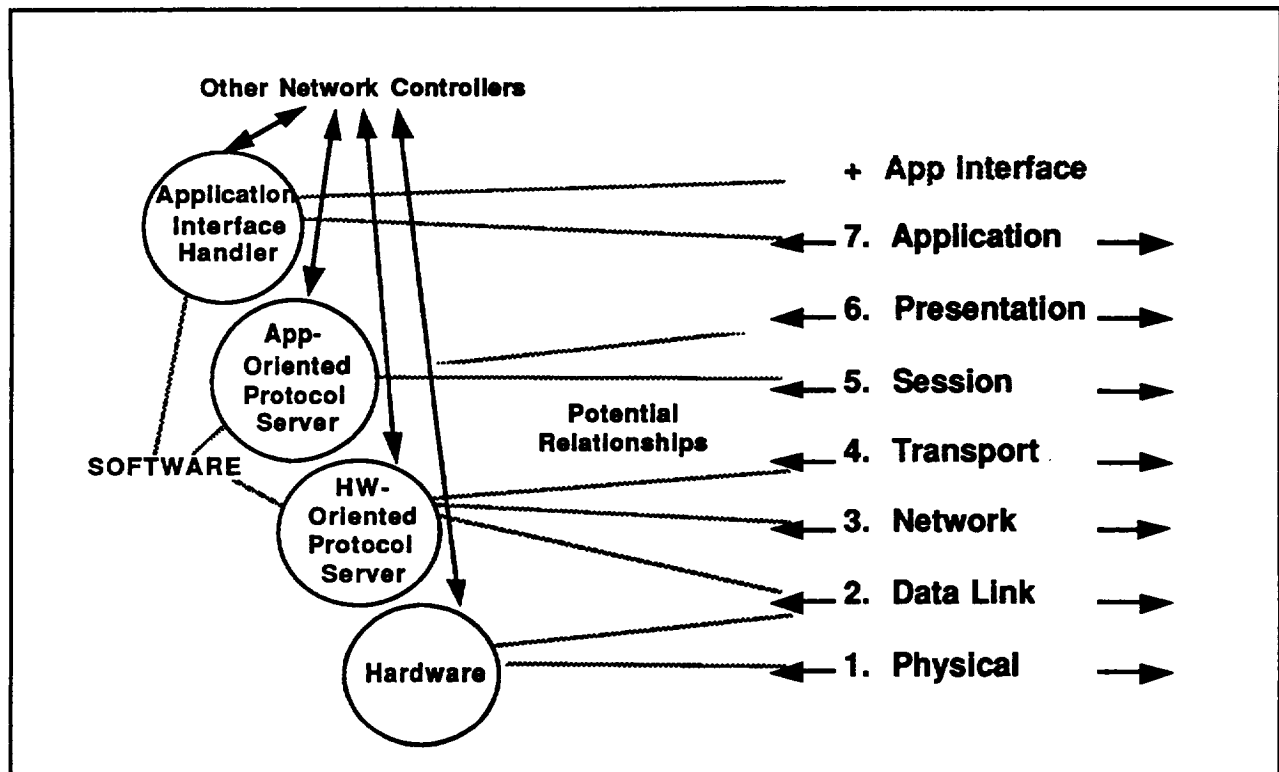


Figure 3-8. Stack Software and Hardware Partitioning

### 3.4.1.3.1 Network Service Controller

The Network Service Controller (NSC) as shown in Figure 3-9 is the element in the NSM that provides all user access to the network. It is the controlling function for the NSM. All user access to the network is to be through a NSM Interface Definition (NSMID). This NSMID will consist of a set of service primitives for each NSM function.

All requests for network service and responses to those requests for services are through the NSC prior to being sent to the appropriate NSM function for implementation. The NSC provides for command verification, program management (Network Stack Control), management of the queue of service requests by priority designation, management of time for the complete SDSS, and is the interface to applications using the NSM. The NSC exposes all of the NSMID services to users.

In order to provide a standard interface to the network and isolate application/user uniqueness, all requests for services to the NSC and responses to those requests from the NSC will be through the Standard Data Service Manager and not direct to the applications/users. Network health and status data is provided to the Data System Manager

for evaluation and appropriate action as required. Health and status data is also provided to the Network Manager for use in reconfiguration of the network to accommodate failures.

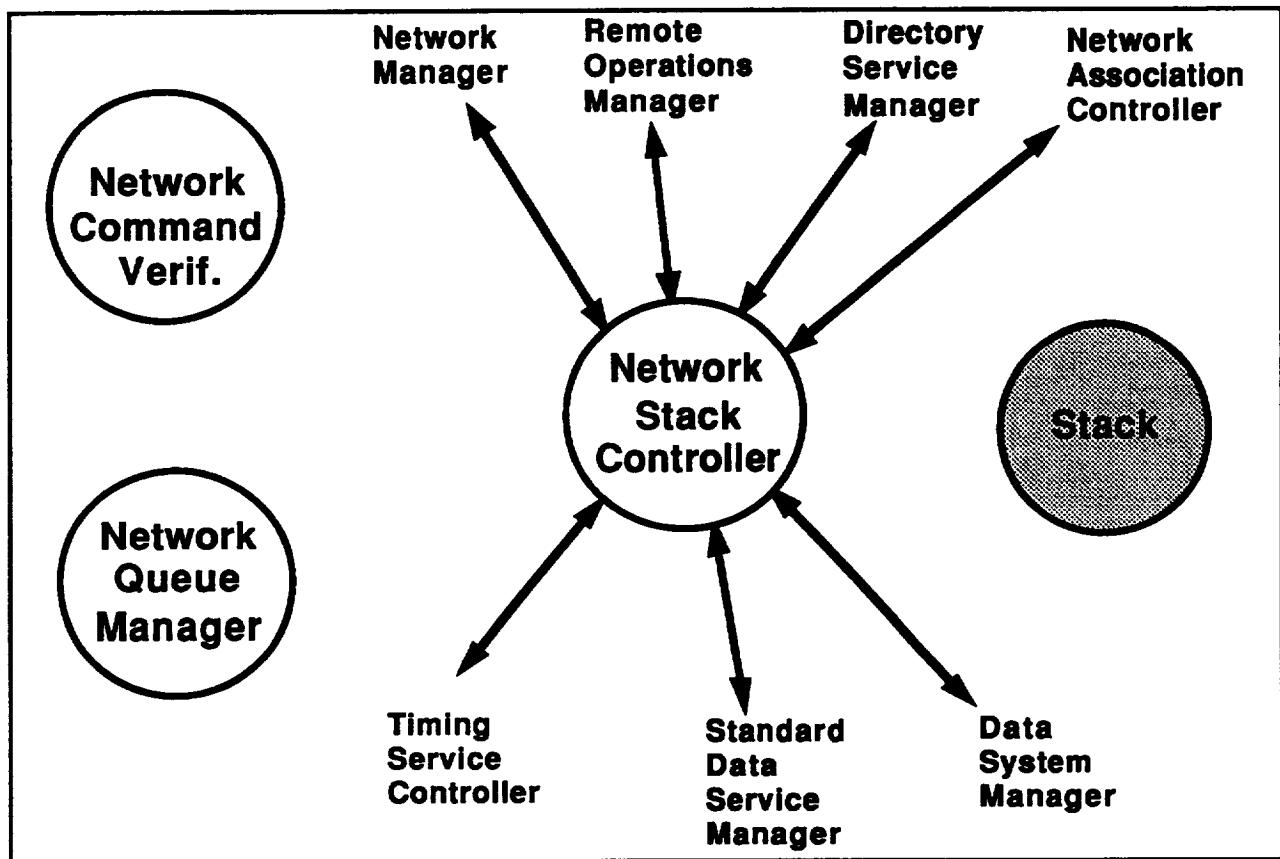


Figure 3-9. Network Service Controller

#### 3.4.1.3.2 Network Manager

Network Manager (NM) functions can be divided into network coordination (configuration management), performance management, security management and network fault detection, isolation and control (FDIR) management. Network coordination accumulates, develops and maintains a network address table that contains the addresses of all processing elements on the network. Performance management gathers data for performance analysis and accounting. Performance management data is sent to the Data System Manager for evaluation and initiation of action to resolve performance problems that may occur. In specific instances, the network may reconfigure without commands from the Data system Manager to bypass hard failures. Security management detects unauthorized attempts to enter the network or modify network parameters and reports infractions to the Data System

Manager. FDIR Management collects and reports permanent errors, transient errors and failed attempts to establish a peer-to-peer association to the Data System Manager. FDIR Management performs authorized reconfiguration in the event of a hard failure.

#### **3.4.1.3.3 Remote Operations Manager**

The Remote Operations Manager (ROM) is the entity that provides the capability for interactive applications processing in an open systems environment. A remote operation is defined as one in which an application in one processing element requests an operation by an application in another processing element on the network. The ROM establishes the protocol necessary to request, control and acquire the results of remote operations.

#### **3.4.1.3.4 Network Association Controller**

This entity provides a capability for two application entities to establish, maintain, control and release an application association over the network.

#### **3.4.1.3.5 Directory Service Manager**

The NSM Directory Service Manager (NSM DSM) provides a directory of names, addresses and other information needed by the NSM to establish communications over the network. The NSM DSM contains the protocol required to obtain access to the directory and the names and addresses of all processing elements on the network. In addition, the NSM DSM will contain the names and status of all application entities to which an association may be attempted. The NSM DSM protocol will provide a capability to obtain access to directory services on external networks. NSM DSM will provide the capability to be modified by the Data System Manager in order to reflect network and application status. Requests for service from remote directories will use the capabilities of the Remote Operations Manager.

#### **3.4.1.4 Data System Manager**

The Data System Manager (DSM) shown in Figure 3-10 is the system executive, the entity that manages and monitors the Space Data System hardware and software. The DSM executes under the Operating System. In a distributed processing system the DSM will have a single processing element designated as the DSM with parts of the DSM functions distributed to the other processing elements on the network. For fault recovery purposes, at least one additional processing element on the network should be assigned as a redundant

"hot backup" to the DSM processor. The distribution of functions is necessary to provide the DSM with the ability to acquire the data needed to manage the system. In a centralized processing system all DSM functions will be in the central processing element with a redundant backup being an option dependent on system requirements.

The DSM performs initialization, startup, configuration and reconfiguration of the data system, maintains the configuration, monitors data system health and status, handles assigned data system FDIR responsibilities and manages the time distribution system.

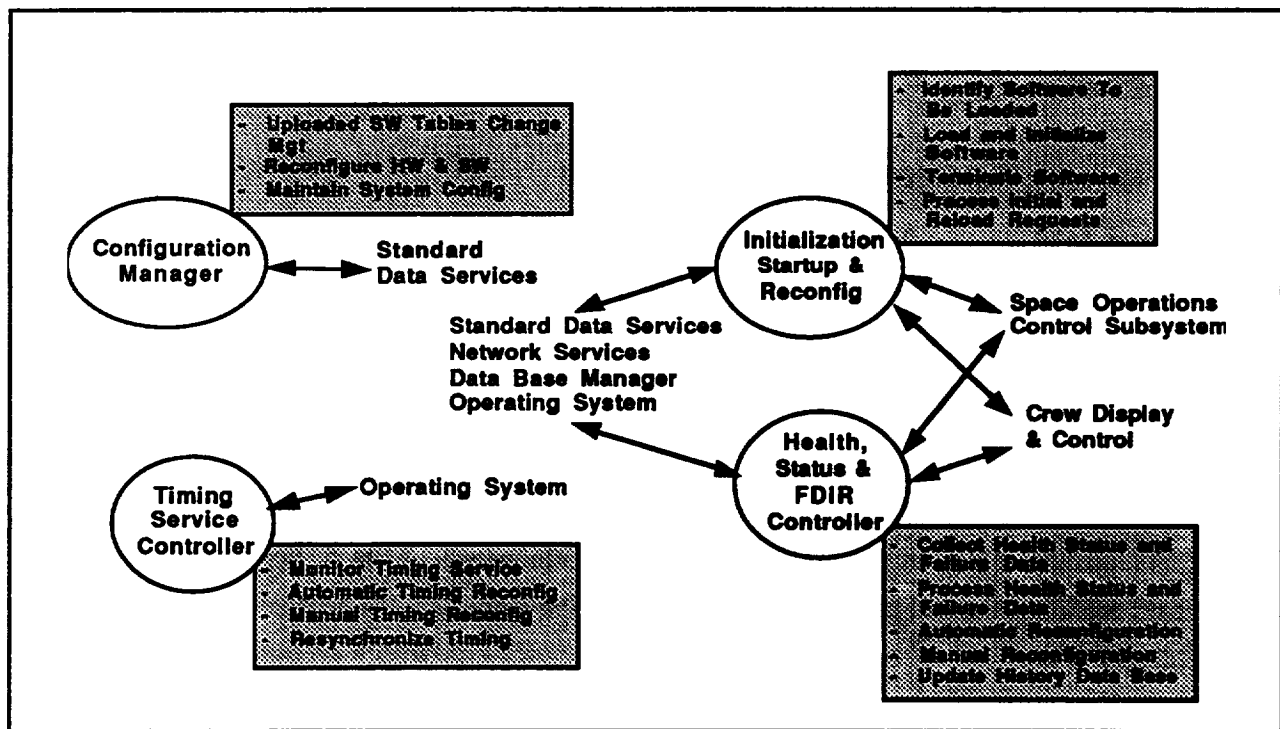


Figure 3-10. Data System Manager

#### 3.4.1.4.1 Configuration Management

This function maintains the system configuration data base. Prior to the mission, the initial system configuration is defined and loaded into the data base. This system configuration table will contain the topology of the system, processing element addresses, application software load allocations, service software load allocations, fault recovery reconfigurations processes and all other data needed to completely define the data system. During operation, DSM will use this configuration table to manage the data system. DSM is also responsible

for updating this table to account for all reconfiguration activity. The crew or SOCS through the DSM can initiate changes to the configuration.

#### **3.4.1.4.2 Initialization, Startup and Reconfiguration**

At power-on, the capability shall be provided for the DSM processor to load the DSM software, perform self-test, broadcast that it is operational and take command of the system. Simultaneously, the backup DSM processor will load the backup DSM software and broadcast that it is operational. In the event the backup DSM processor does not receive a message from the DSM processor that it is operational and has assumed command, the backup DSM processor will interrogate the DSM processor. If a satisfactory answer is not received from the DSM processor by the backup DSM processor, the backup DSM processor will command the DSM processor off-line, designate another processor as backup and take command. If a satisfactory answer is received by the backup DSM processor from the prime processor, initialization will continue in a normal sequence. A similar sequence shall be followed in the event the backup DSM processor does not initialize to an operational state.

The following sequence of events shall follow initialization of the DSM. On a data system wide basis, the DSM Initialization, Startup and Reconfiguration entity is responsible for accessing the system configuration table at system initialization to determine what software is to be loaded into the various processing elements and in what sequence. Once that determination is made, the DSM shall load the appropriate software into each processing element and initialize the processing elements in the predefined sequence. Each processing element shall perform self-test as a part of the initialization sequence and report test results to the DSM Health, Status and FDIR Controller. Initialization failures shall be handled in accordance with the pre-loaded fault handling and reconfiguration tables. Successful completion of the initialization sequence shall be reported to the SOCS and Crew and the DSM Health, Status and FDIR Controller shall begin gathering status data on all system elements. In the event that the required status is not reached within a specified time period a fault condition shall be reported.

This DSM entity can also be requested within mission guidelines by the crew, SOCS, a pre-planned sequence of events, applications through Standard Data Services, Network Services, Operating System or the Health, Status and FDIR Controller to load and initialize software, terminate software and reconfigure the data system.

Within each processing element, the capability must be provided upon DSM command to load, initiate and terminate application software via the Operating System. The capability must also be provided to download application software and issue system configuration commands to processing elements attached to local communications.

#### **3.4.1.4.3 Health, Status and FDIR Controller**

This controller consists of a four stage process. Each processing element gathers health and status, then processes the data if capable, sends the data to the DSM which then processes the data from all sources to perform Health, Status and FDIR control on a data system wide basis. All detected faults and failures are reported to the crew and to SOCS. The degree of automatic recovery and reconfiguration based upon detected faults and failures is dependent upon the system implementation and the mission.

In each processing element, the Health and Status (H&S) function collects the application processor H&S data. Faults are reported to the application processor local FDIR function. All H&S data along with fault data is also sent to the DSM H&S and FDIR Controller. The application processor FDIR function processes the H&S data to determine the health and status of the processor. This function alerts the crew, SOCS and the DSM of detected faults, failures and overloads.

The system H&S function resides in the DSM processor. It collects H&S data from all processing elements and maintains this information in a data base. The data is also sent to the system FDIR function for processing to detect faults and failures. H&S is responsible for maintaining a fault log for all system components and for controlling the reconfiguration of the system resources. This function provides for automatic reconfigurations to account for detected faults and failures, based on pre-defined configurations and mission rules contained in the system configuration table. This function also provides for manual reconfigurations in response to crew and/or SOCS commands. The recovery process for each fault subject to automatic recovery must be defined in the system configuration table. This function also detects system overloads. All detected faults and failures are reported to the crew and SOCS.

#### **3.4.1.4.4 Timing Service Controller**

This function monitors the status of the time source and the time distribution service. It also coordinates the distribution of time to all processing elements. It provides the

capability for automatic reconfiguration within established system configuration table guidelines based upon faults or failures. It also provides the capabilities for manual reconfiguration and resynchronization based upon commands from the crew and/or SOCS.

### 3.4.1.5 Operating System

The Operating System (OS) as shown in Figure 3-11 provides the layer of SDSS software that isolates other services as well as application software from the data processing hardware element. The OS provides management, allocation, and deallocation of the processor, memory, timing and input/output (I/O) processing resources for application and service software. The OS architecture for the SDSS provides for custom Ada software applications (Ada RTE), and commercial off-the-shelf (COTS) utilities and applications (OS Kernel). In addition, OS/RTE Extensions is provided to accommodate device and resource management functions unique to the SDSS that are not normally provided by an Ada RTE. Non-standard processor services are provided to accommodate unique mission processing requirements. Note that the Space Station OS/Ada RTE has many features common to this generic OS architecture.

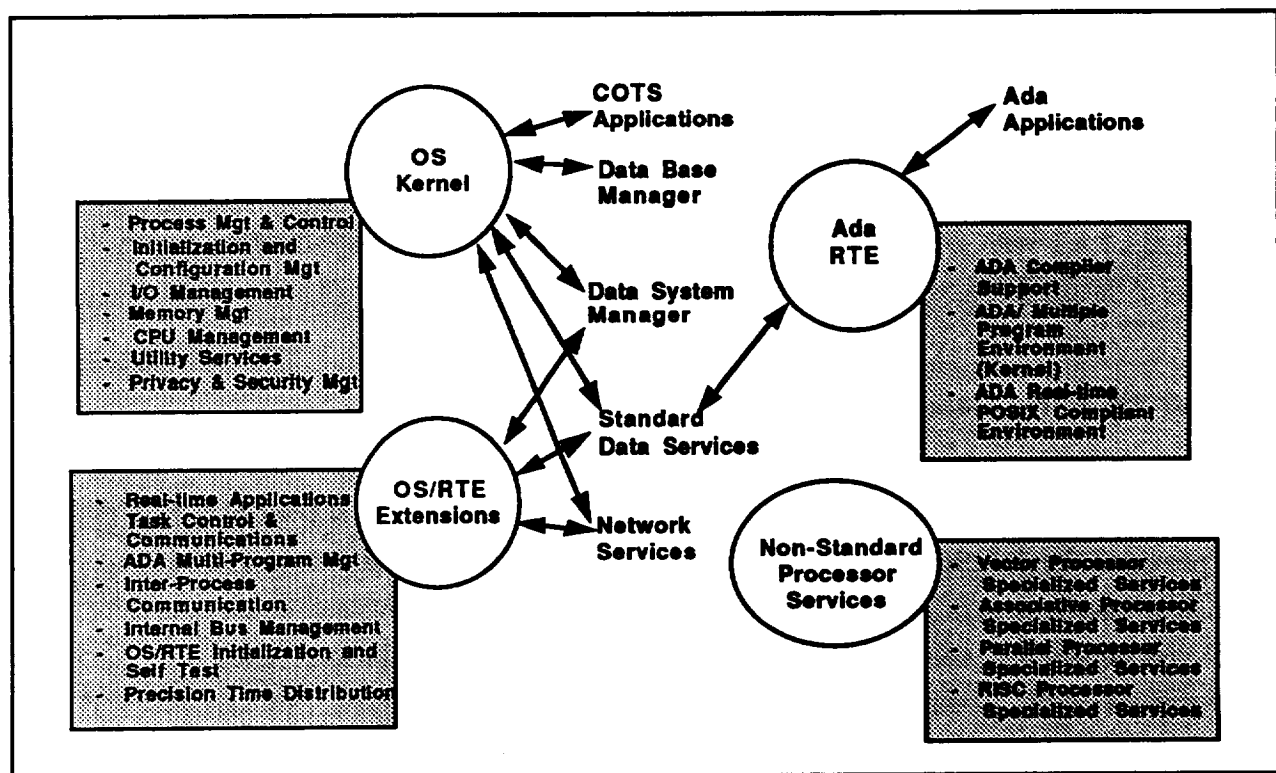


Figure 3-11. Operating System Services

#### **3.4.1.5.1 OS Kernel**

The OS Kernel provides an operating environment which supports COTS applications, existing non-Ada compatible COTS software and Ada applications through the OS/RTE Extensions by compliance with pre-defined API standards. API standards should provide for transition to or be POSIX compliant. The OS should be compatible with the POSIX OSE Reference Model as described in reference [POSIX91] and OS Kernels used in commercial processors. Major functions provided are process management and communications, memory management, I/O operations, CPU management, privacy and security, software initialization and configuration management, utility services and provision of software error monitor and logging support for FDIR. The process management function should provide a multi-program environment enabling one or more multi-tasking real-time Ada application programs to run simultaneously with multiple non-real-time programs.

#### **3.4.1.5.2 Ada RTE**

The Ada RTE entity is to provide the necessary processing services to support the Ada programming language as defined by ANSI/MIL-STD 1815 (latest version), Reference Manual for the Ada Programming language. The Ada RTE will also support and be supported by the OS/RTE extensions and be supported by a POSIX compliant OS. Processing services to be provided are task management and communication, memory management and I/O management. The Ada RTE will be required to operate in two environments. The first environment is a characterized by multiple Ada programs with multi-tasking supported by a POSIX compliant OS. The second environment is characterized by multiple real-time Ada programs implemented on an OS kernel.

#### **3.4.1.5.3 OS/RTE Extensions**

OS/RTE Extensions provides those functions required by the SDSS that commercial OS Kernels or Ada RTEs do not normally provide. It is to be noted that if the OS Kernel or Ada RTE provides the function it does not have to be developed as an extension. The extensions can be separated into the two categories. The first consists of those functions which are dependent upon the hardware and software implementation. They are management of the backplane bus internal to the processing element, initialization and self-test, and timing distribution. The second category consists of those functions which have the potential for compatibility with POSIX Ada bindings (IEEE P1003.5) as the POSIX



standard continues to be developed. These functions are real-time applications task control and communications, Ada multi-program management and inter-process communication.

#### **3.4.1.5.4 Non-Standard Processing Services**

This function provides those processing services which are unique to a specific vehicle or mission. A representative list of potential services is shown in Figure 3-11. These services would be provided by a SAP configured for the unique needs of the system.

#### **3.4.1.6 Data Base Manager**

This entity provides services to the SDSS subsystems and application users for the management of structured data files, file transfers and file redundancy management. In a distributed processing environment with a data processing element or elements assigned to function as a mass storage device and containing multiple processing elements on a network, the generic architecture would be as shown in Figure 3-12. For a centralized processing system with only one processing element (not including redundancy) the functions shown to transfer data files between processing elements (nodes) on a network would not be required. The Data Base Manager (DBM) executes under and uses the services of the OS. All communication with and requests for services from the DBM are through the Standard Data Services Manager.

##### **3.4.1.6.1 File Service Controller**

The function provides the services necessary to create and manage structured files. Files conforming to a standard fixed structure are baselined in order to provide the capability for file transfers between non-homogeneous software systems. The File Service Controller (FSC) responds to requests for file transfers from the Distributed file Transfer Controller function located in other nodes. This function also manages the DBM resources such as driver tables and provides for file redundancy based on a pre-defined data criticality.

##### **3.4.1.6.2 Node Directory**

The DBM will be provided a directory at system initialization of node names and addresses authorized for specific file transfers. The DBM will perform file transfers in accordance with this directory. The directory can be modified by the crew or SOCS with the DBM being responsible for maintaining the directory.

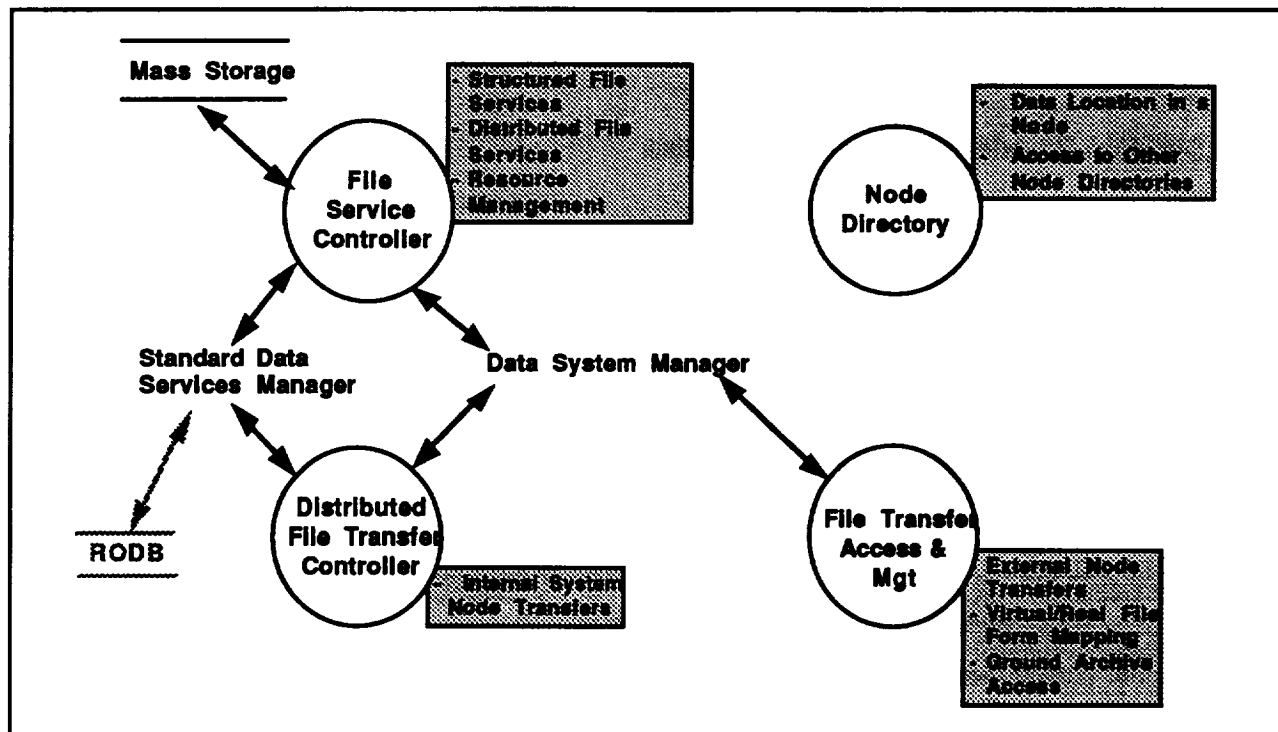


Figure 3-12. Data Base Manager

#### 3.4.1.6.3 File Transfer Access and Management

This function provides the means to transfer structured files outside the SDSS to systems with heterogeneous software. The capability to receive files from heterogeneous systems will also be provided. The file will be converted to virtual file store form and transferred in this form upon receipt of an authorized request. The receiving system must provide the capability to convert the virtual file store form to the receiving system real file store form. The DBM will provide the capability to convert from virtual file store form to SDSS real file store form.

#### 3.4.1.6.4 Distributed File Transfer Controller

This function provides the means to transfer structured files inside the SDSS upon request from applications, the crew or SOCS to the requested location.

### **3.4.2 SPACE OPERATIONS CONTROL SUBSYSTEM**

In the SGOAA , the SOCS is the high level integrating command and control functional entity for a space vehicle and mission. SOCS functions may be allocated to both ground mission control facilities and onboard space vehicle facilities. As illustrated in Figure 3-13, the Crew, through the SDSS, has direct interface to and ultimate control of all vehicle subsystems through the various controller entities. The crew also has a direct manual interface option to the subsystems through the Systems Controller. Ground control functions are implemented through the Command Controller. For unmanned vehicles, the Command Controller is the primary control source. The Systems Controller is the direct interface to all subsystems and implements the commands of each individual controller entity. The Systems Controller, Vehicle Controller and Command Controller are discussed in more detail in the following paragraphs.

Although these vehicle operations control functions may be partitioned and allocated to some degree between ground and space-borne control facilities, this does not imply there are no other functions to be performed by a ground control facility. The focus in SOCS is on vehicle operations control, not overall mission control, which may imply other functions will be needed to be performed in a ground control facility.

#### **3.4.2.1 Vehicle Controller**

The function of the SOCS Vehicle Control Manager is to coordinate the actions of the individual subsystem controllers with regard to overall operation of the vehicle. Figure 3-14 illustrates some of the major functions that this manager must perform. Vehicle mode is a function of mission phase and/or vehicle situation. Each mode requires a different set of control rules and commands. Regardless of vehicle mode, the Situation Awareness Manager has the responsibility for being aware of vehicle state with regard to the external environment and taking action or providing alerts as appropriate to ensure mission success and prevent vehicle or crew loss. The State/Attitude Controller has the responsibility to know internal vehicle states and to coordinate these states with incoming commands to change states in order to implement the incoming commands in a safe and optimum manner. The Alternative Response Manager has the responsibility to determine and maintain awareness of vehicle capabilities and provide this information as needed to authorized requesters to include recommending changes to commanded actions if the vehicle capabilities will not support that action.

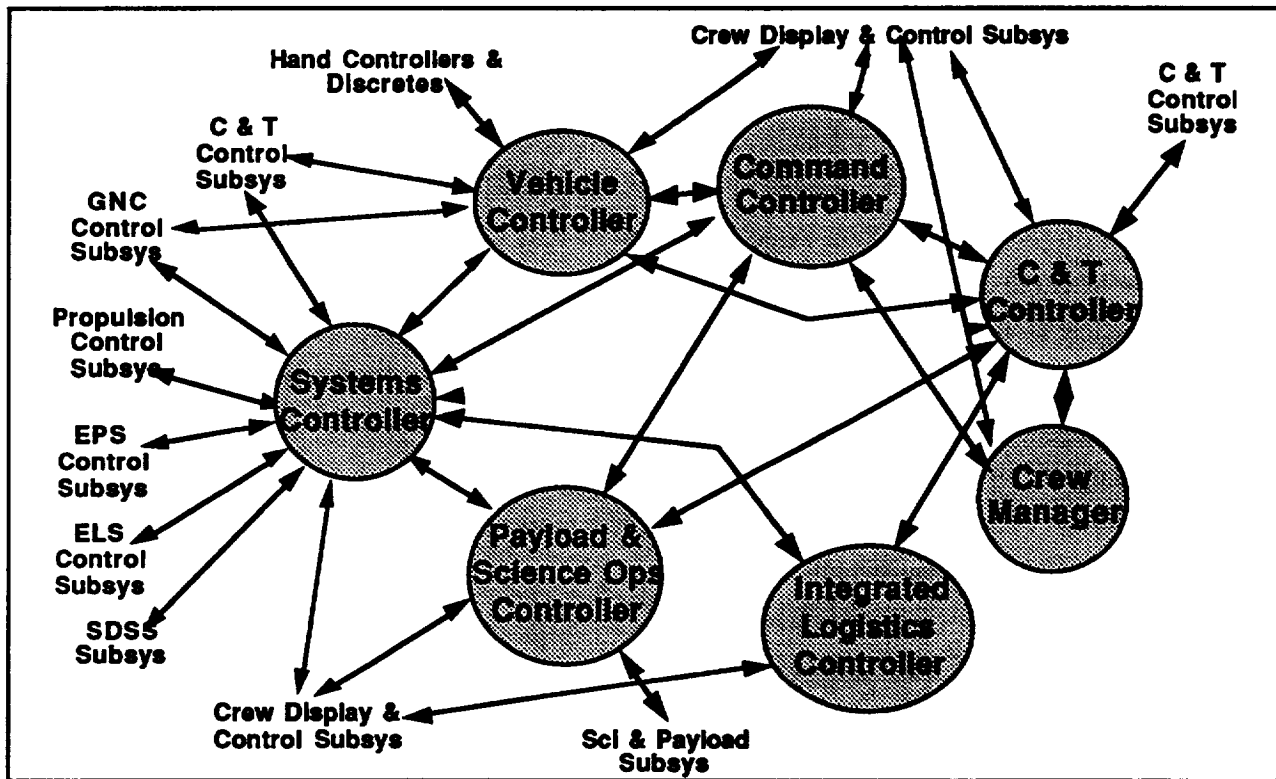


Figure 3-13. Space Operations Control Subsystems

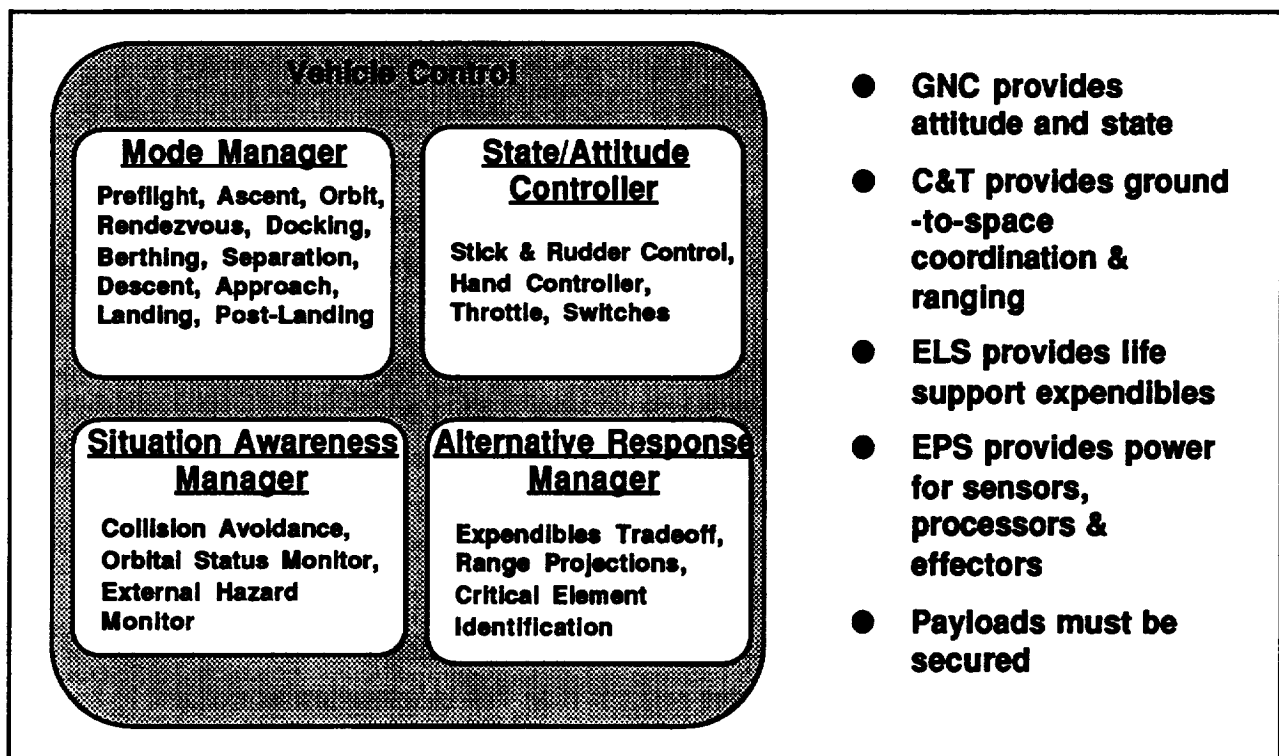


Figure 3-14. Subsystem Coordination through Vehicle Control

#### **3.4.2.2 Command Controller**

This controller is the central control entity for the spacecraft and mission. The functions, as shown in Figure 3-15, are allocated to onboard implementation or to the ground control center, based upon vehicle and/or mission need. In a manned vehicle a capability is provided for crew override. The Command Manager provides a capability to implement those command and control sequences necessary for specific mission accomplishment. The Mission Plans Manager is the primary builder and keeper of the plans (from both the current mission and all older missions that preceded the currently underway mission), and is the entity that develops predefined timelines or adapts them to real-time commands from external authorized sources such as the crew or ground control personnel. The Mission Operations Manager is the execution entity for the mission timelines or real-time commands. The Mission Information Manager is responsible for gathering mission activities, operations and subsystems data and making this data available to users. Each of these functional entities can be allocated to either the ground mission control center, the mission director, the spacecraft command processing subsystem, or the spacecraft commander. They can be allocated to the backroom support agencies, the offboard operations control or the onboard operations control. This definition of Command Control enable the architecture to insure all operations control requirements are identified, allocated to implementation facilities, and include pre-planned interface requirements.

#### **3.4.2.3 Systems Controller**

The Systems Controller is the vehicle controller and is the command and control interface to all vehicle subsystem controllers from all other control entities, as shown in Figure 3-16. This entity coordinates the interaction of all vehicle subsystems in responding to command inputs from all sources in order to enable the vehicle to safely and effectively accomplish the mission. The System Controller must at all times be aware of vehicle state and resource status and implement all commands received in accordance with the impact on vehicle state and capability. In manned vehicles, the crew has a direct interface to issue commands to subsystems through the Systems Controller. The crew will also have a capability to override Systems Controller commands under predefined conditions. Each subsystem will have an internal control entity for its own operation.

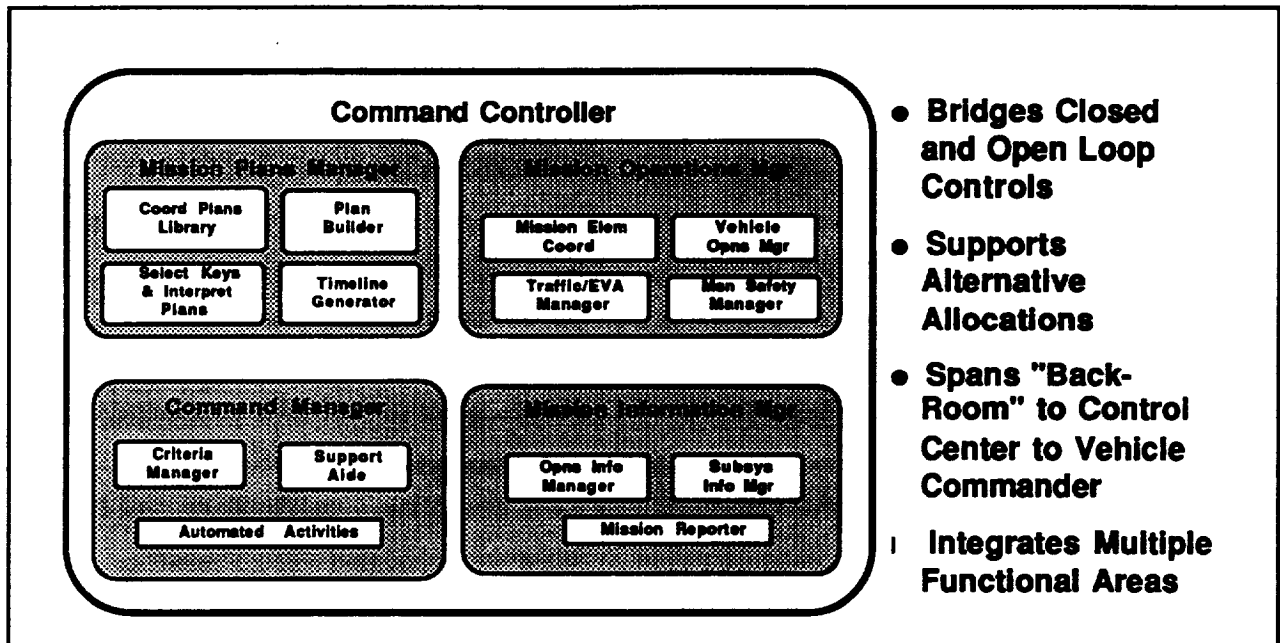


Figure 3-15. Needed Functions in Operations Command Control

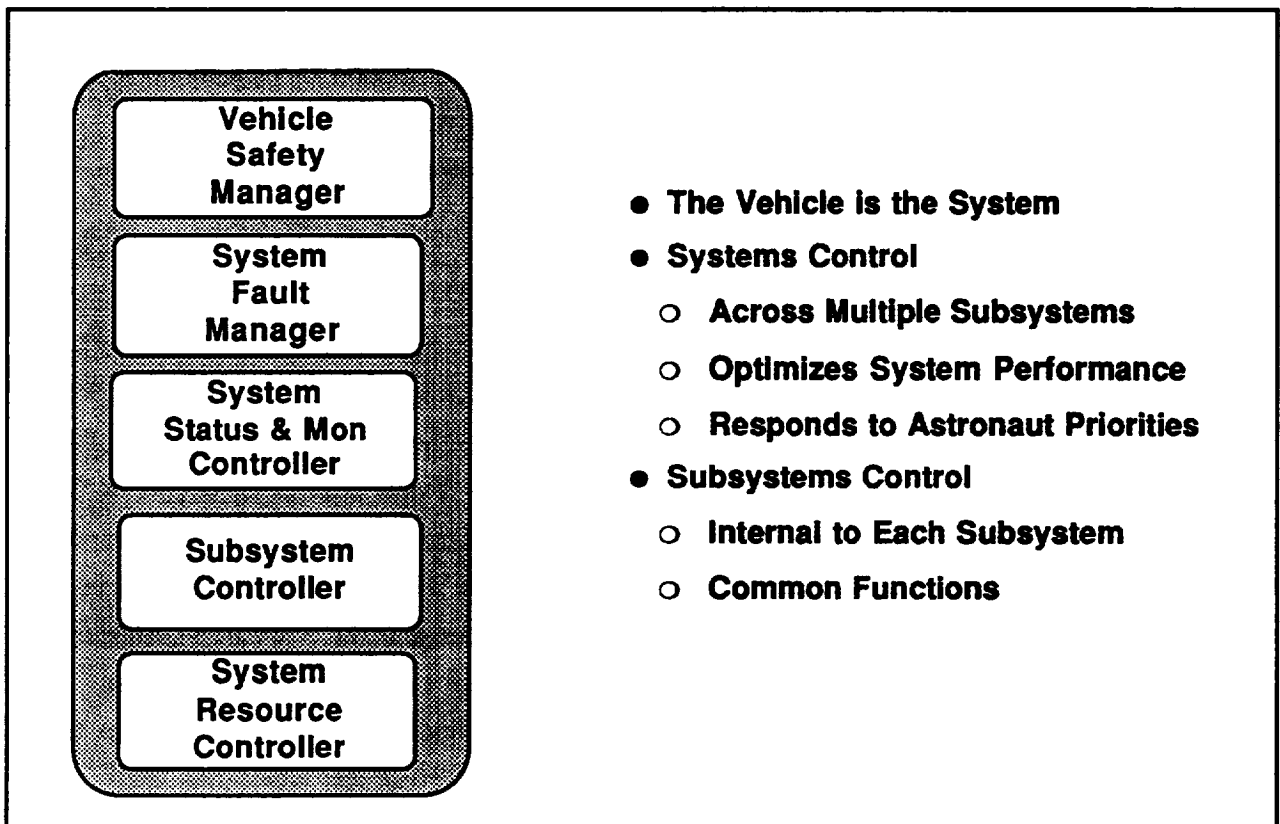


Figure 3-16. Systems Control Optimizes Functionality

#### **3.4.2.4 Communications & Tracking Control**

The Communications and Tracking Control application prepares the communications directions to implement command control function guidelines, maintains communications configuration data, and manages tracking configurations to enable the crew to maintain control over communications subsystem software.

#### **3.4.2.5 Crew Manager**

The Crew Manager application is the overall management applications system for supporting the vehicle crew activities. It consists of scheduling software to monitor crew time schedules, medical applications to monitor crew health and checkups, and training applications and simulations to enable the crew to perform training while in flight.

#### **3.4.2.6 Integrated Logistics Control**

The Integrated Logistics Control subsystem performs the logistics management and the maintenance management functions for the spacecraft. This function is shown in Figure 3-17. It insures that logistics support elements are available when needed to support vehicle and crew activities, and that maintenance can be performed as needed to insure all components on the spacecraft will be operable when needed. Maintenance addresses both planning activities, preplanned routine repairs and ad hoc emergency repairs as needed.

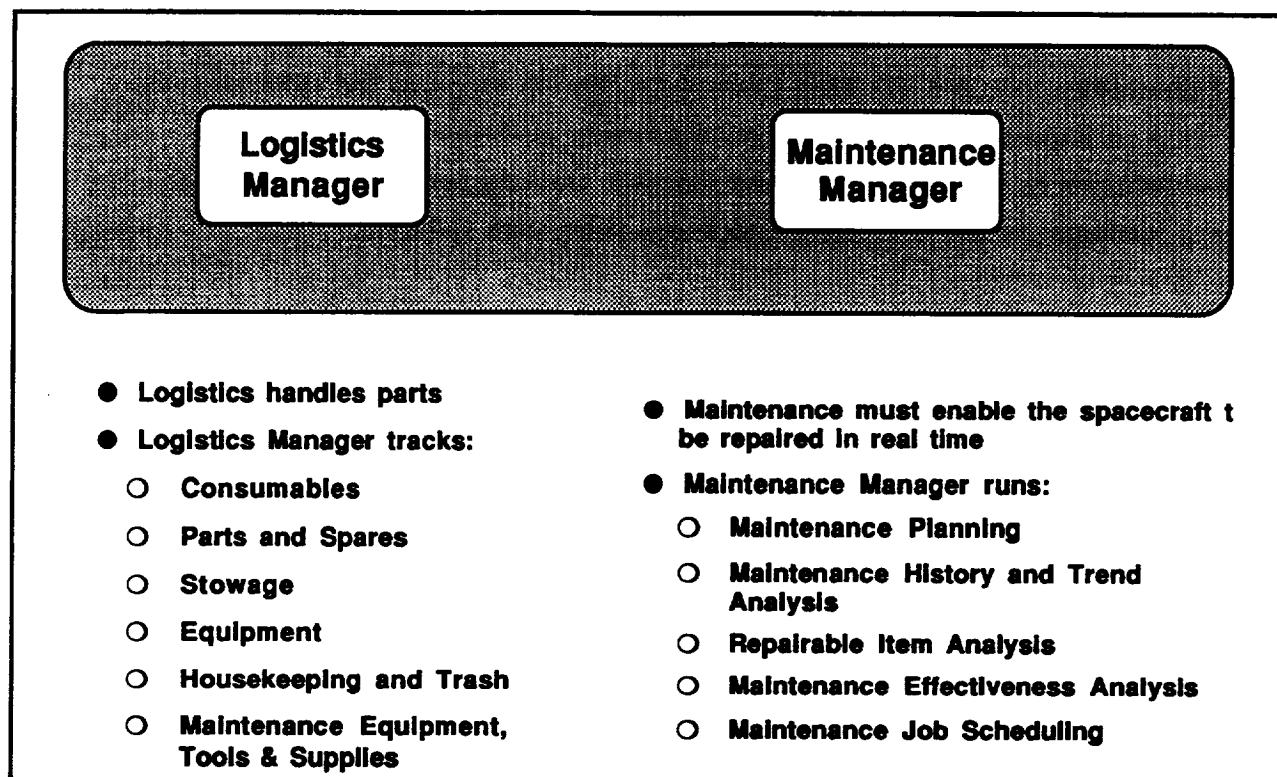


Figure 3-17. Integrated Logistics Control Supplies and Fixes Broken Spacecraft

#### **3.4.2.7 Payload and Science Operations Control**

The Payload and Science Operations Control subsystem is a microcosm of the overall SOCS application, tailored to the needs of individual payloads. It performs the functions shown in Figure 3-18. The payload and science command control function provides the command processing for the payload and its command interface to the rest of the spacecraft. The payload and science systems control performs the health and status monitoring and control for the payloads and their interface to the spacecraft subsystems. The payload and science vehicle control function provides a control subsystem for real-time and non-real time operation of remote spacecraft and rovers. The payload and science communications control provides the control interface between the SOCS communications controller, the avionics communications subsystem and the payload communications subsystem.

The payload operations control subsystem architecture is structured to provide maximum (potential) independence to the payload subsystems so they may operate independently of the vehicle operations control. Since the payloads may have missions aspects different from the host vehicle with concurrently performed missions, which may not be linked, this will provide greater flexibility to the operation of payloads and science missions.



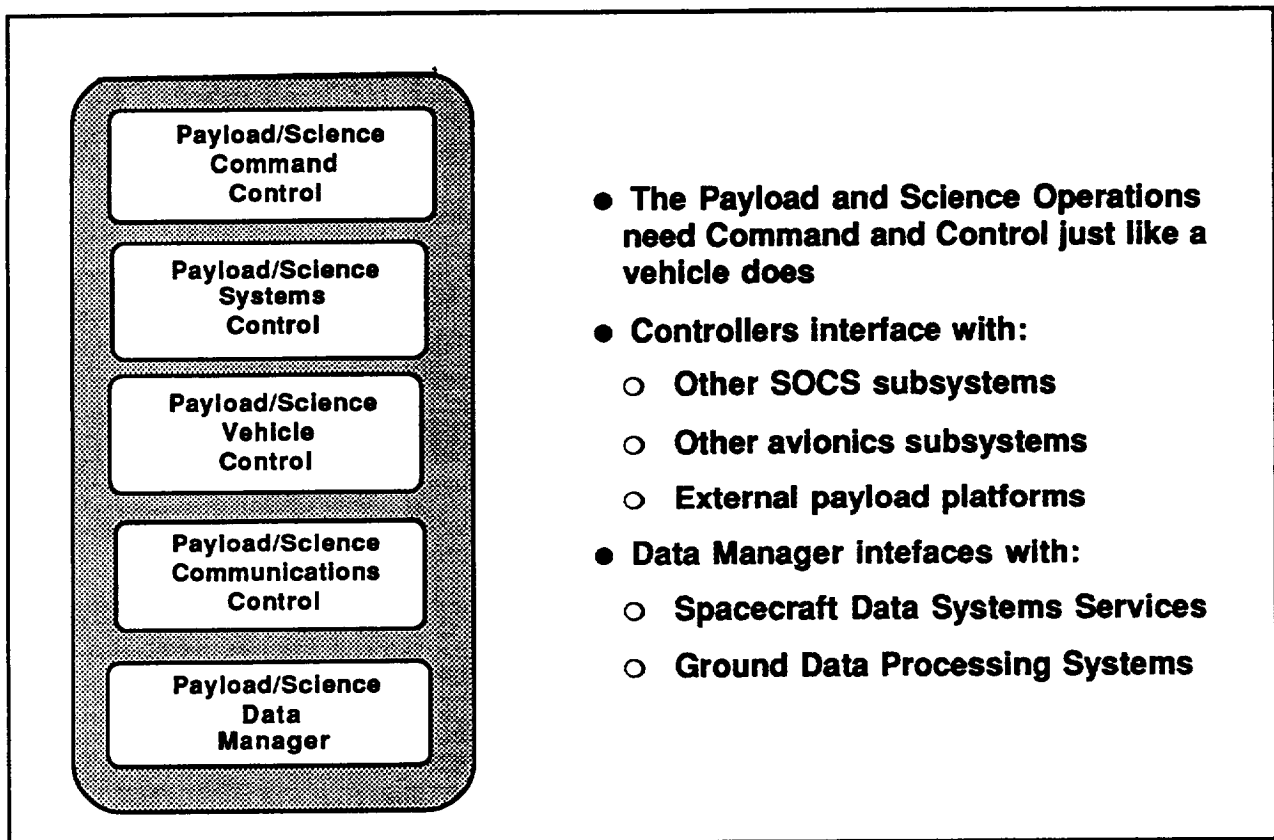


Figure 3-18. Payload and Science Operations Control



### **3.5 SPACE DATA SYSTEM ARCHITECTURE APPLICATIONS**

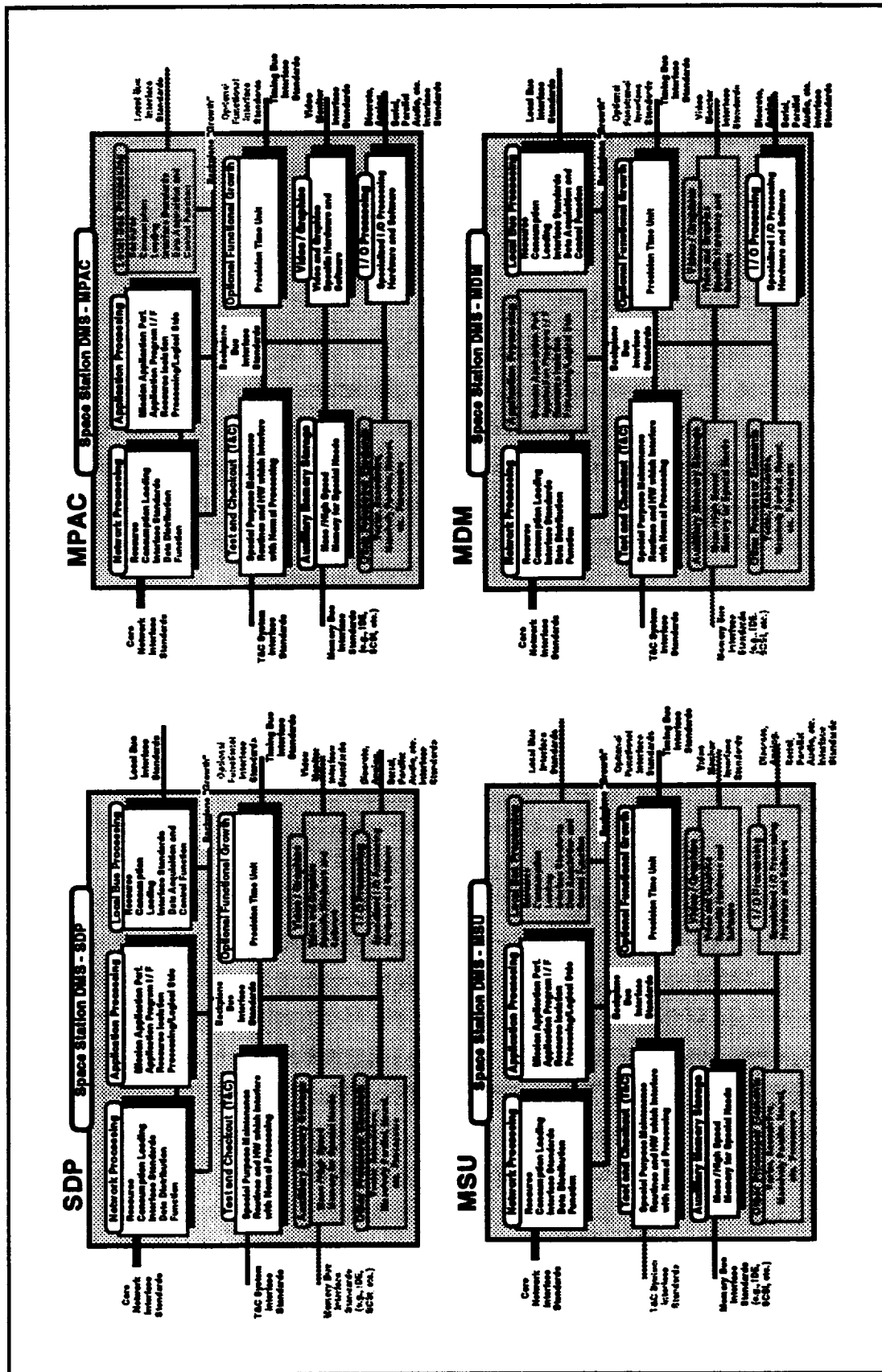
The SGOAA can be used to implement avionics architectures for specific mission requirements, and to define the detailed functions and subfunctions and the preferred partitioning between hardware and software. This section describes a comparison of the SGOAA to the Space Station architecture, and describes how the SGOAA was used to develop the preliminary architecture requirements for the Common Lunar Lander.

#### **3.5.1 SPACE STATION APPLICATION**

In order to test the validity of the Generic Processing Hardware Architecture Model, shown previously in Figure 2-16, a test case was conducted to determine if the functions provided by this architecture model were capable of satisfying the Space Station DMS Processor functions. The results of this test case are shown in Figure 3-19, with each of the four small block diagrams representing Figure 2-16. The equivalency of the Generic Processing Hardware Architecture functions to the Space Station DMS Hardware Architecture functions are shown below:

<b><u>Generic Processing Architecture</u></b>		<b><u>Space Station DMS Architecture</u></b>
• Network Processing	=	Network Interface Unit
• Application Processing	=	Embedded Data Processor
• Local Bus Processing	=	Bus Interface Unit
• Test and Checkout	=	Software Development and Diagnostic Unit
• Auxiliary Memory Storage	=	Mass Storage Device
• Video/Graphics	=	Video/Graphics
• I/O Processing	=	Direct Interfaces to Sensors and Effectors

As can be seen from Figure 3-19, all of the DMS functions are satisfied. For example, the Space Station Mass Storage Unit (MSU) requires the Network Processing function to communicate over the FDDI network. The Application Processing Function is required to handle assigned Data Base Management and Data System Management System Services Processing. The Auxiliary Memory Storage Function is required as the basic function of the MSU is to provide mass data storage for the Space Station. The Test and Checkout Function is required for hardware diagnostics as well as software development, test and diagnostics. The Standard Data Processor (SDP), Multi-Purpose Application Processor (MPAC) and the Multiplexer/Demultiplexer (MDM) functional requirements are similarly accommodated. The Time Generation Unit and Base-band Signal Processor are considered to be SAP units and as such require unique functional configurations.



Note: For text inside each module, see Figure 2-16, Generic Processing Internal Hardware Architecture Model.

Figure 3-19. Generic Avionics Architecture vs. Space Station Design

### **3.5.2 COMMON LUNAR LANDER APPLICATION**

The modularity and tailorability of the SGOAA was tested by applying it to the avionics design of a new initiative vehicle called the Common Lunar Lander. The elements of the SGOAA presented below were tailored to the preliminary Common Lunar Lander system requirements in order to define the data management and command system requirements.

The Common Lunar Lander (CLL) is an unmanned lander designed to land a 60 Kg payload anywhere on the surface of the Moon. It will be launched on an Expendable Launch Vehicle and has a nominal mission duration of 5 days. The conceptual design includes most of the major subsystems that one would expect of any space vehicle including: Guidance Navigation and Control, Communications, Tracking, Power, Propulsion, and of course a Data System. The only interface to the payload is a structural attachment (i.e. no data, power etc.).

#### **3.5.2.1 Space Operations Control Subsystem Requirements Tailoring**

Figure 3-20 presents the SOCS elements after tailoring to the CLL requirements. The shaded areas represent those elements which are not required in order to implement the CLL design. For example, since the CLL is unmanned, there is no need for the Crew Manager Function. The Payload and Science Operations Control function was eliminated because the only interface to the payload is the physical attachment. Likewise the Integrated Logistics Control function was eliminated because of no requirement for these functions on board the CLL.

While the above functions were deleted in their entirety, some of the other functions were tailored at a lower level. For instance, when allocating the vehicle control function, all elements were eliminated except the mode control functions.

One of the guidelines was to keep the vehicle as simple as possible. Since this is an unmanned vehicle, failures are not life threatening. For this and cost reasons, single string design concepts were emphasized if part reliability could be estimated to provide a reasonable chance of mission success. Thus for safety, fault tolerance and reliability issues, this meant not using elements of the architecture which were available. For the Command Control and System Control functions, those elements related to safety, fault tolerance and reliability were deleted. Similar criteria were used to delete other elements of the Command Control and System Control functions.

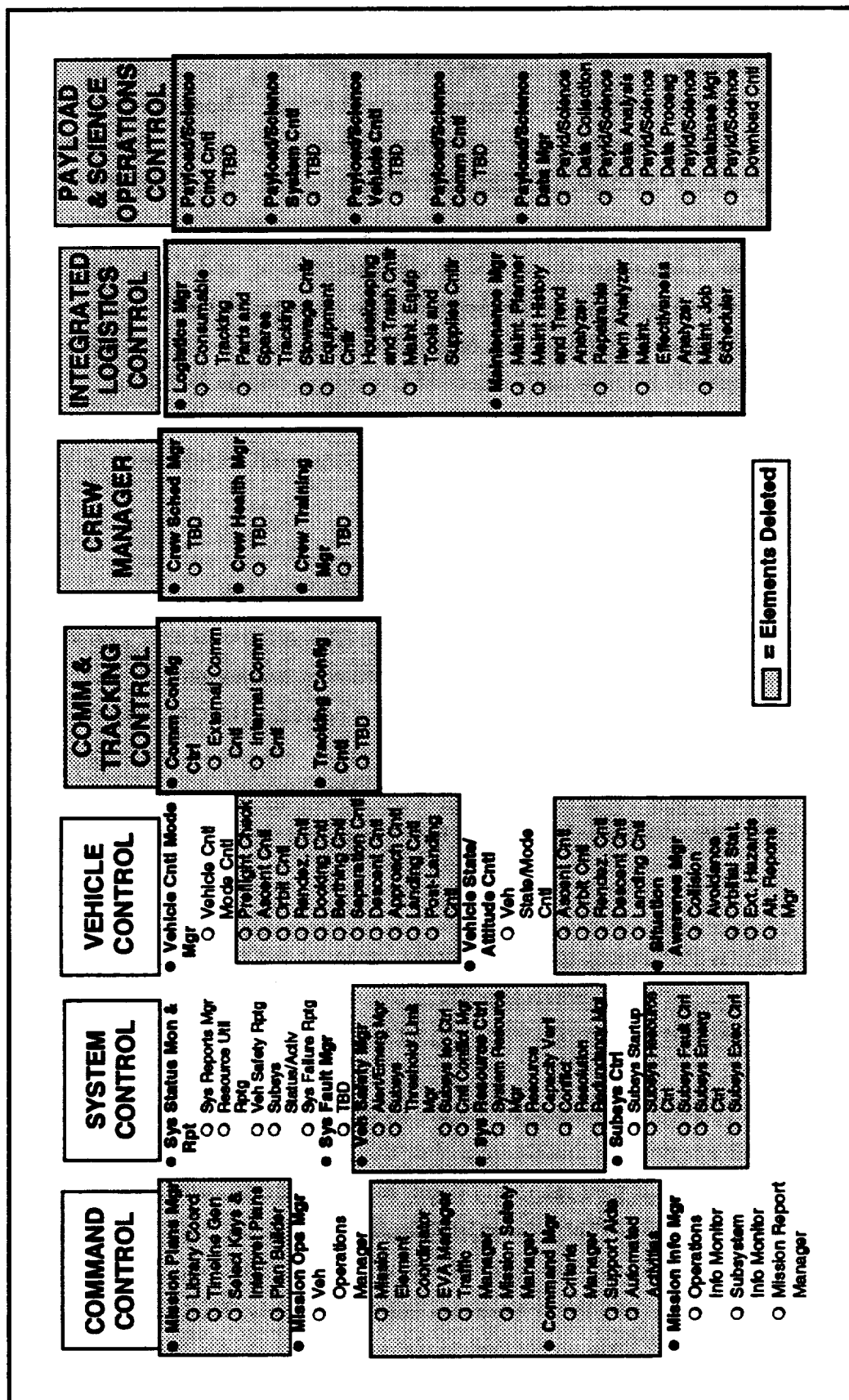


Figure 3-20. Space Operations Control Subsystem Requirements Tailoring

### **3.5.2.2 Space Data System Services Requirements Tailoring**

Figure 3-21 presents the SDSS elements after tailoring to the CLL requirements. There is no requirement for a core network in the Common Lunar Lander data system concept. This is because, for the vehicle size and mission, the data system reduces to one node, thus there are no peer-to-peer communications taking place. For this reason, the Network Services Manager was deleted.

The other deleted elements are fairly self explanatory. The hashed-out area under Operating System represents tailoring that will take place in the future. At the time this tailoring was done, it was not clear as to whether an Operating System Kernel would be required or an Ada Run Time Environment. It is clear, however, that both are not required, so the tailoring will take place at some future time when the requirements become more defined.

### **3.5.2.3 Space Data System Hardware Requirements Tailoring**

As mentioned above, there is no requirement for a core network in the Common Lunar Lander data system. Thus communications between subsystems will be of the Local Communications variety. This leads to the Space Data System Hardware Tailoring presented in Figure 3-22. The grayed out area in the upper half of the figure shows the SGOAA hardware not required for the Common Lunar Lander mission. Other subsystems are represented as instantiations of the GAP(M), Sensors or Effectors.

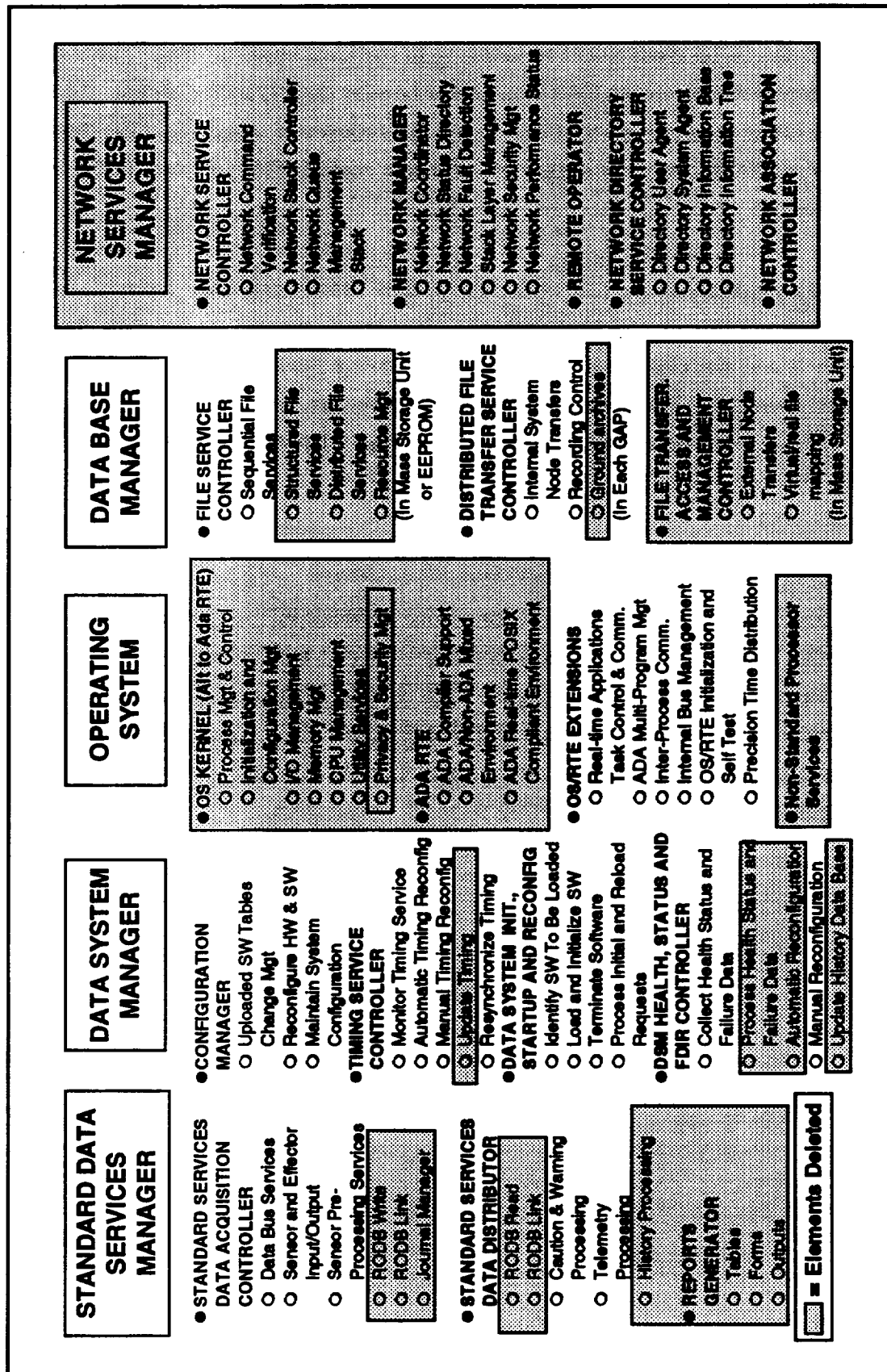


Figure 3-21. Space Data System Services Requirements Tailoring



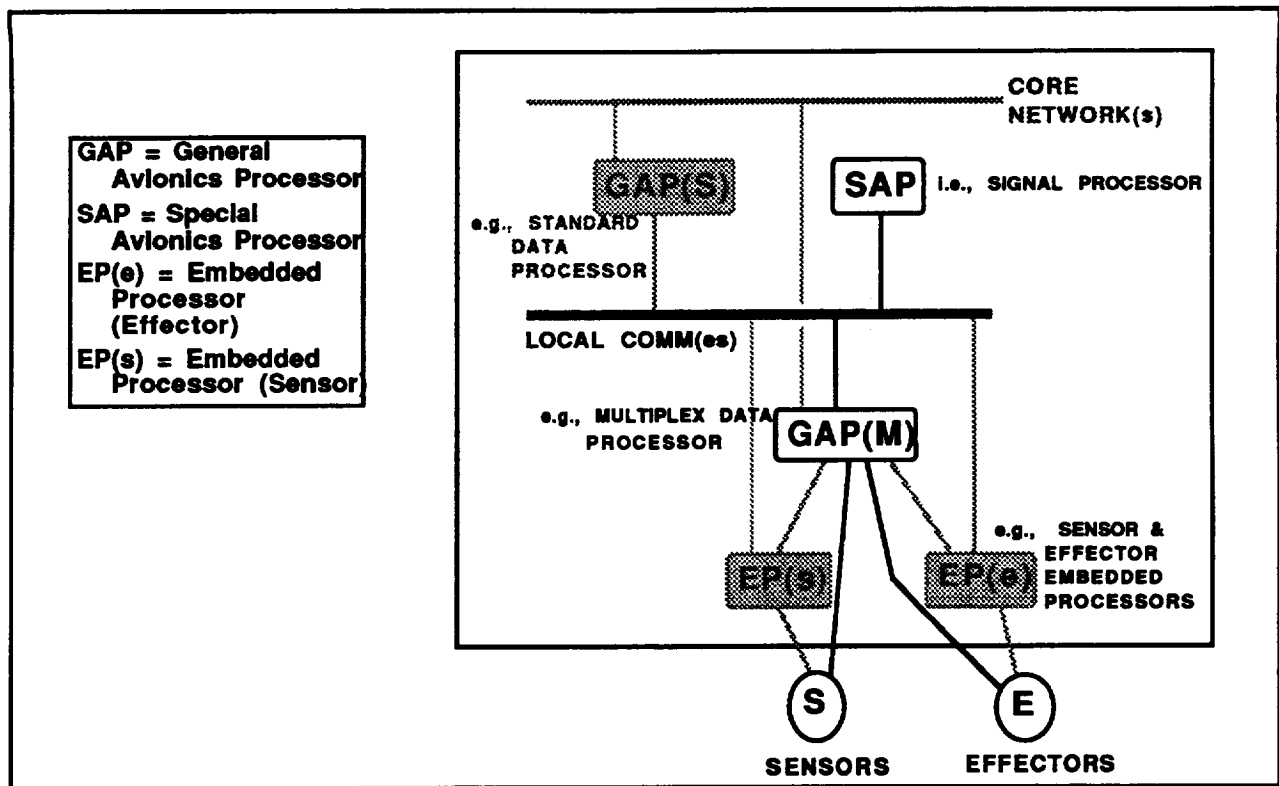


Figure 3-22. Space Data System Hardware Requirements Tailoring

#### 3.5.2.4 GAP Internal Architecture Requirements Tailoring

Figure 3-23 presents the Generic Processing Hardware Architecture Tailoring. The grayed out areas represent the elements of the SGOAA Hardware Architecture that were removed from this implementation since they were not needed. Network Processing was eliminated because there is no core network for the Common Lunar Lander. Ancillary processing elements as well as Auxiliary Memory Storage were not required, and obviously because it is unmanned, video and graphics support was not required. The GAP will be responsible for applications processing, so that function was retained. Likewise, Local Bus Processing is an important feature of the CLL conceptual design. One of the driving requirements for the CLL hardware is the ability to handle various I/O interfaces. It requires numerous analog and discrete interfaces to other subsystems, and thus the I/O Processing function plays an important role in the GAP Internal Architectural. The Test and Checkout function is also required for the CLL.

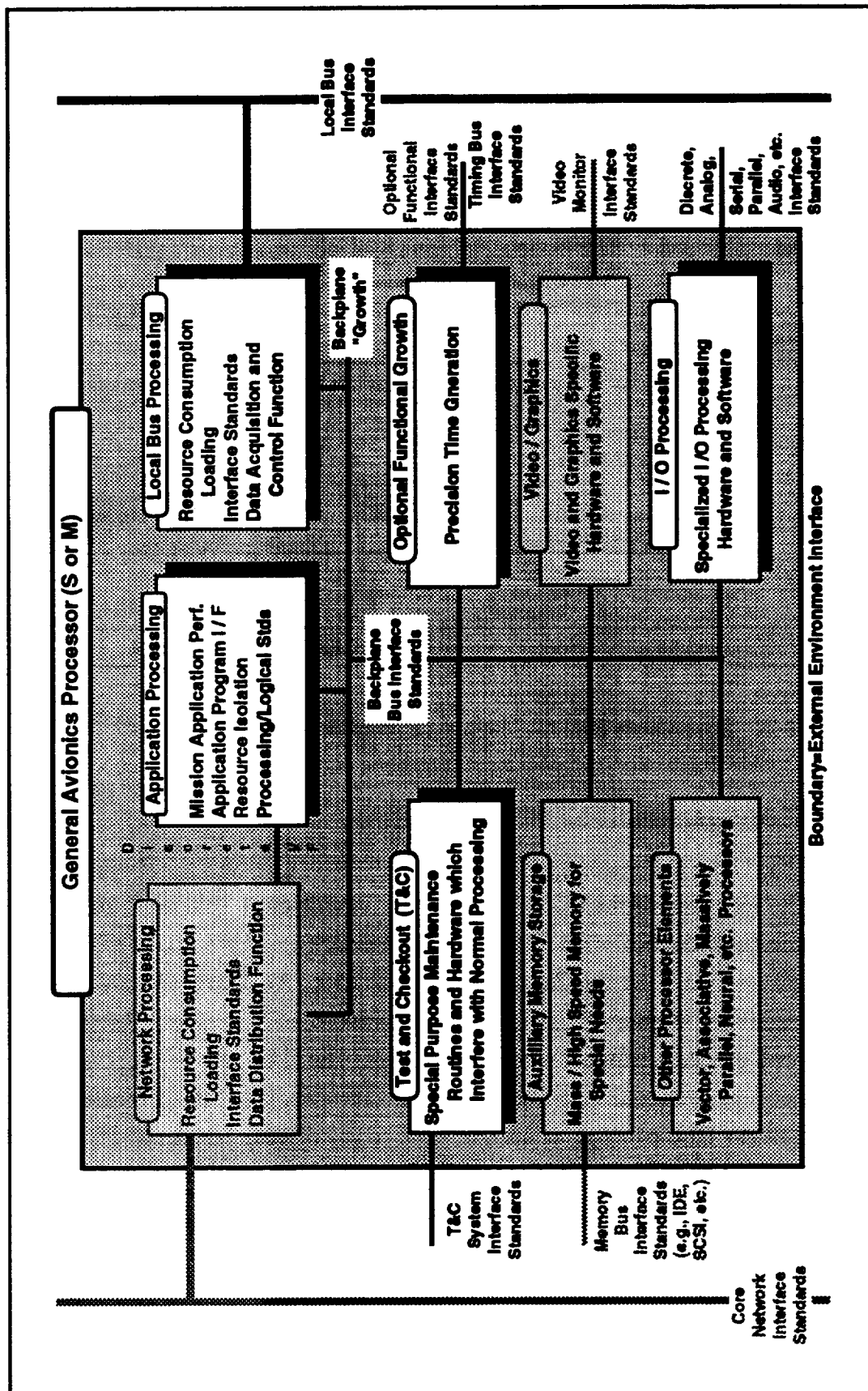


Figure 3-23. GAP Internal Architecture Hardware Requirements Tailoring

### **3.5.2.5 Lessons Learned**

The resulting tailored architecture for the Common Lunar Lander is shown in Figure 3-24.

One of the important issues in the Common Lunar Lander Design was the method of producing, distributing and updating mission time. There were two basic concepts which were considered. First, use a separate Time Generation Unit and a separate timing bus. From a SGOAA point of view this would be analogous to a sensor and thus fits comfortably within the architecture. The other method would be to have the timing unit on a card inside the GAP. Although this option is not explicitly identified as one of the GAP Internal elements, provision for such extensions are allowed by the "Backplane Expansion" capability, and could easily be incorporated as another element attached to the backplane bus. This demonstrates the robustness of the architecture, because new elements can be added in a modular fashion without changing the structure of the architecture.

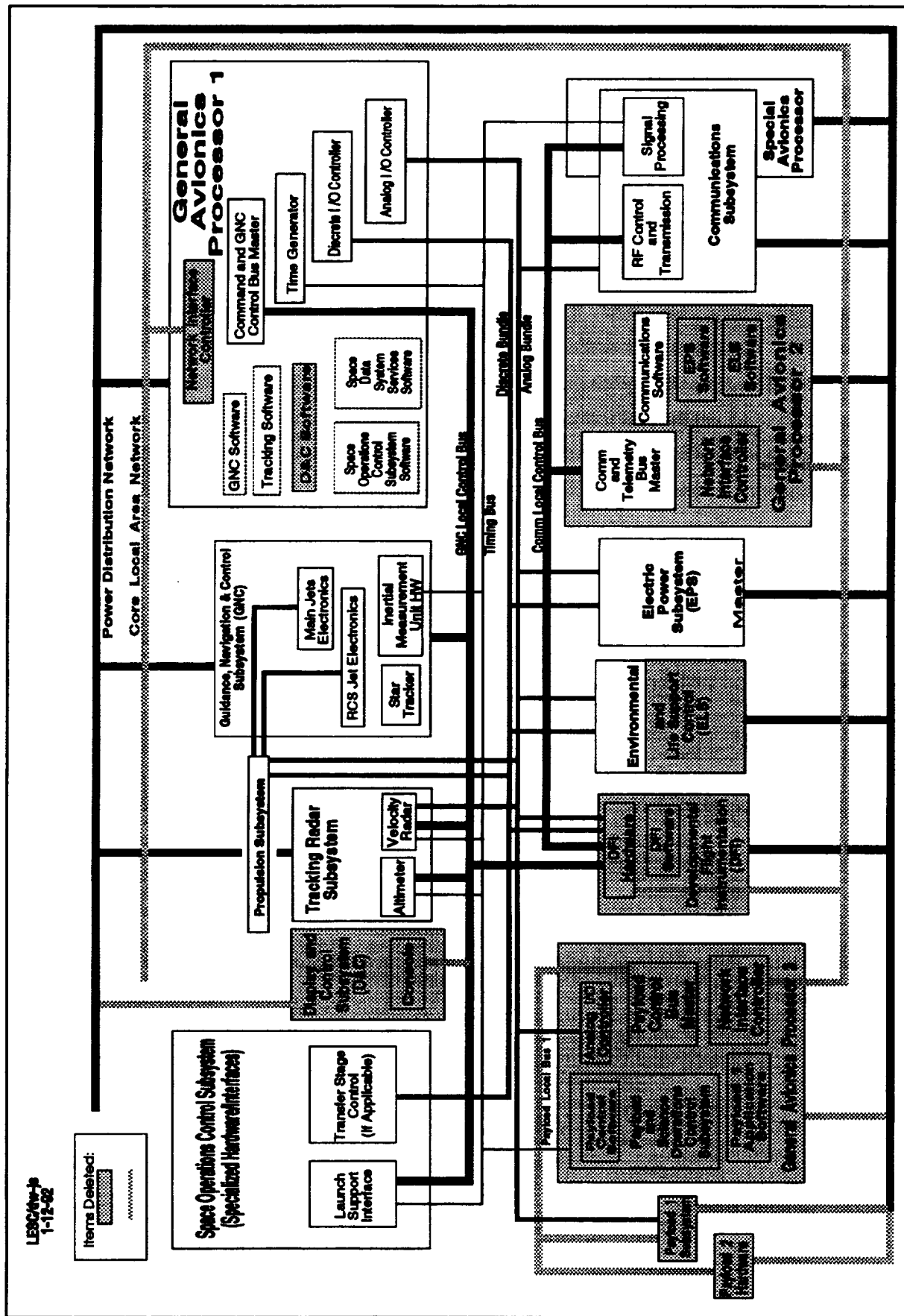


Figure 3-24. Common Lunar Lander Tailored Architecture

## 4. CONCLUSION AND RECOMMENDATIONS

### 4.1 CONCLUSIONS

The SGOAA architecture presented in this report is the product of the continuing "Flight Data Systems Architecture Development and Analysis Task". Updates on the architectural development will be provided as reports and presentations to future SATWG and SAE forums. During this study effort, several guidelines were developed that should be applied to all spacecraft generic avionics architecture development activities. These guidelines are:

- The architecture must be based on standards
- The architecture must be general enough to span platforms for all missions and operational requirements
- The architecture should be a *requirements* architecture; i.e. one that can be tailored for design implementation based on actual system requirements.
- The architecture must be adaptable to varying system requirements.
- An avionics control structure must be integrated into an architecture.
- The architecture must be adaptable to alternate system design and development approaches.
  - Static and Dynamic Analysis Techniques
  - Functional, Hardware, Object-Oriented, Structured Analysis, etc. Methodologies.

The generic avionics architecture discussed in this report has been applied to the preliminary design of the data system for the Common Lunar Lander (ARTEMIS) project. In general, the SDSS architecture was well suited to handle the ARTEMIS requirements. Preliminary evaluation of the design effort to determine lessons learned for application to the continuance of the SGOAA study yielded that the SDSS architecture should be extended to encompass the software development environment architecture and the test environment architecture. The SOCS architecture requires modification to more distinctly define the interfaces and partition the architecture between the spacecraft and ground control.

This application of the SGOAA to the Common Lunar Lander (CLL) enabled performance of a preliminary assessment of the CLL Data System requirements based on CLL System requirements. The preliminary architecture for the CLL Data System was developed from the SGOAA generic SDSS and SOCS detailed functional architectures in approximately two

days. Further work involved a refinement of the preliminary architecture and interviews with other subsystems to determine their requirements on the CLL Data System. This led to the estimation of the overall CLL Data System size and complexity. This exercise successfully validated the usefulness of the SGOAA because it allowed for the realistic estimation of CLL Data System requirements in a very short time. At the same time, lessons learned from applying the architecture such as a GAP Internal Architecture Timing Element will be worked back into the SGOAA.

## **4.2 RECOMMENDATIONS**

This document is presented to share information with SATWG members from the various NASA centers and with participating members from industry in order to solicit their feedback and support in the further development and refinement of this architecture.

It is recommended that the SATWG formally adopt the SGOAA and the following four SGOAA models for space avionics application:

- SGOAA System Architecture Model Definition, as discussed in paragraph 2.3.1.
- SGOAA Architecture Interface Model Classes, as discussed in paragraph 2.3.2.
- SGOAA Generic Processing External Hardware Architecture Model Definition, as discussed in paragraph 2.3.2.1.2.
- SGOAA Generic Processing Internal Hardware Model Definition, as discussed in paragraph 2.3.2.1.3

In addition, the following recommendations are made for extensions to the ongoing NASA JSC Flight Data System Divisions SGOAA study:

- Incorporate results/recommendations of relevant SATWG previously-commissioned architecture studies into the SGOAA.
- Extend the SGOAA to include development of a Software Architecture model.
- Extend the SGOAA to include development of a System Development and Test Environment model and interfaces.
- For the interfaces identified in the SGOAA, determine the standards requiring development and their requirements.

- Extend the functional architectures supporting the SGOAA by completing the open functional architecture for the SOCS and developing one for the human oriented D&C subsystems.
- Apply the SGOAA to the design of a future Avionics System such as the "First Lunar Outpost" and run simulations of the design using dynamic analysis tools and techniques to validate the resultant design concept, evaluate system performance against requirements and to test potential SGOAA interface and requirements interactions.

It is also recommended that the SATWG initiate development of generic avionics architectures for the following avionics subsystems that are outside the auspices of the NASA JSC Flight Data Systems Division.

- Electrical Power Control Subsystem
- Environmental and Life Support Subsystem
- Payload Operations Control Subsystem
- GN&C Control Subsystem
- C&T Control Subsystem





**APPENDIX A**  
**SGOAA DEFINITIONS**



## **APPENDIX A**

### **SGOAA DEFINITIONS**

Abstraction is the principle of using only those aspects of an entity, object, operation, function, process, or other subject which are relevant to the current purpose and ignoring those aspects not needed to improve analytical focus on the current subject. This principle simplifies a complex subject to render it more susceptible to analysis and design. In object oriented approaches, data abstraction is the principle of defining a data type in term of the operations that apply to the entities of the type.

Application is the use of capabilities (services/functions) provided by an information system specific to the satisfaction of a set of user requirements. (POSIX P1003.0 Draft 14 Guide)

Application Platform (AP) is the set of resources that supports the services on which an application or application software will run. Also known as a host platform. (POSIX P1003.0 Draft 14 Guide)

The application platform provides services at its interfaces that, as much as possible make the specific characteristics of the platform transparent to the application.

The Application Program Interface (API) is the interface between the applications software and the applications platform, across which all services are provided. (POSIX P1003.0 Draft 14 Guide)

The API is primarily in support of application portability, but system and application interoperability are also supported by the communications API.

Application Software is software that is specific to an application and is composed of programs, data and documentation. Application software has uniquely defined interfaces. (POSIX P1003.0 Draft 14 Guide)

Architecture is the structure of Application Software, API, AP, and EEIs which describe the organization and interfaces of a system.

Avionics System is the set of all electronic and processing based subsystems on a space vehicle, including all hardware, software and other electronics needed to control and operate the space vehicle. It is the collection of system elements and allocated capabilities that provides the coordinated functionality for end-to-end processing in handling the information needed to interface the space vehicle's major components, to control its

interaction with its environment, and to respond to human commands. (Adapted from [JSC 31000])

Communication Interface is the boundary between application software and the external environment, such as application software on other host platforms, external data transport facilities and devices. The communications interface may be internal to one space vehicle or across multiple space vehicles. (POSIX P1003.0 Draft 14 Guide)

The services provided are those whose protocol state, syntax and format all must be standardized for interoperability.

Communications components include phone lines, global networks, local area networks, and packet switching equipment.

Component is one of the parts resulting when an entity is decomposed into constituent parts.

Concurrent engineering is defined as the application of multiple engineering disciplines to develop requirements in several different but related areas at the same time so the requirements are coordinated and mutually supportive.

Continuity is defined to mean that requirements changes are proportional to design changes, i.e., that changes in the requirements will propagate into changes of the same order of magnitude in the design.

Control Subsystem is an application which selects and implements alternative actions based on a-priori criteria or real time guidance..

Core Avionics is defined as the control subsystems and the supporting avionics (hardware and software) needed to enable these control subsystems to function. Core avionics include the controls for each of the traditional space avionics hardware subsystems (such as Guidance Navigation and Control (GN&C) and Communications and Tracking (C&T)). The avionics hardware sensors and effectors are outside the core avionics boundary.

Data Base Manager is the control subsystem which manages structured data files, file transfers and file redundancy management.

Data Processing Subsystem is an application subsystem providing data processing services. Data processing subsystems do not perform control subsystem functions.

Data System (for example the Space Data System - SDS) is a network of data system services, onboard computational resources, data storage, and human-machine interface devices which provide onboard command and control, data transmission, computation/processing,

and operating applications software to support a space vehicle's users (crew and controllers), interfacing systems, applications and subsystems.

Data System Services (for example the Space Data System Services - SDSS) is a service subsystem with a generic functional architecture designed to provide a comprehensive set of services to all vehicles and subsystems.

Data System Manager is the control subsystem which manages the housekeeping and status control services for the SDSS, including health management.

Decomposability is defined to mean requirements can be broken into smaller pieces with potentially simpler solutions or at least better understanding and a capability for further decomposition as needed.

Degraded mode is a system condition wherein some system elements (such as hardware, software, human, or procedural) are sufficiently unhealthy that the system cannot operate normally.

Dependability is the integrated capability for reliability, maintainability, fault detection and isolation, reconfiguration, and fault recovery with a non-stop operating system.

Direct Interface is defined as the connection between an entity sending or receiving data with another entity receiving or sending data for transmission of the same data along the routing path associated with moving data from the source of the data to the end user of the data. Data is used by an entity in a direct manner if it passes the data on without changing the data; thus, for example, network operating systems are direct interfaces between applications when they package or unpack data and send it to another network node.

Distributed System is a collection of computers, memories, buses and networks that are concurrently operating in a cooperative manner and communicating with each other.

End-user of data is the last entity which makes a significant transformation, conversion or operation on the data.

Entity is an abstract element that represents an object in the real world, its data attributes and essential services with their respective performance and quality characteristics.

External Environment (EE) is defined as a set of external entities with which the application platform exchanges information. These entities are classified into the general categories of human users, information interchange entities and communication entities. (POSIX P1003.0 Draft 14 Guide)

These entities are classified into the general categories of human users, information interchange entities and communication entities.

External Environment Interface (EEI) is defined as the interface between the application platform and the EE across which information is exchanged. The EEI is defined primarily in support of system and application interoperability. This interface consists of human/computer interaction services, information services, and communications services. (POSIX P1003.0 Draft 14 Guide)

Extensibility is the ability of an architecture to be extended or adapted to new conditions, changes in specifications or new technologies.

Flight Critical Function is a function which, if it fails, could cause loss of vehicle control resulting in loss of the vehicle and crew.

Function is an action/task that the system must perform to satisfy customer and developer needs.

Generic Architecture is an architecture where the elements of the architecture do not depend on any one mission or program for their definition. The elements of a generic architecture can be tailored to apply to many different missions and programs.

A Handler Subsystem is a data process which implements a predefined, directed procedure, either from a control subsystem or a management subsystem.

The Human/Computer Interface is the boundary across which physical interaction between a human being and the application platform take place.

Interface is the shared boundary between two functional units, defined by functional and other characteristics, as appropriate.

Interoperability is the ability of two or more systems to exchange information and to mutually use the information that has been exchanged. (POSIX P1003.0 Draft 14 Guide)

Information hiding (also called encapsulation) is the principle used in developing system structures where components should encapsulate or hide a single requirements or design decision, with an interface that reveals little of the inner workings of the system. Software information hiding refers to the technique of making the external interface to an entity public, but keeping the internal design details hidden from view. Hiding of the internal design information allows the implementation of the entity to be changed without requiring the external interfaces of the entity to be changed. By hiding the internal

implementation, changes are easier to make with minimal rework when system changes are needed.

Information Interchange Components are things like removable disk packs, floppy disks, security badges and remote data bases on other application processors accessed by way of Communications services.

The Information Interchange Interface is the boundary across which external, persistent storage and data interchange services are provided.

The physical format, data format, code sets and data descriptions are required to be specified for data portability and interoperability.

Inheritance is the principle of receiving properties or characteristics from an ancestor. In architecture definition, it allows the specification of common attributes and services only once because they can then be passed to all descendent or referenced subsystem architectures or elements.

Laboratory Architecture is defined as a structure which is capable of being configured to represent a subject open system architecture. Thus, it must include (but not be limited to) non-proprietary standard communications, processing and interfaces. Interfaces to a simulation of the subject's operational environment is included. The lab architecture must include instrumentation, benchmarks, test/simulation controls, displays, and data analysis capabilities. It must be extensible through the addition of subsystems, services and resources following published rules. It must be precisely described and maintained.

Logical Interface is defined as the requirements associated with establishing a data interchange interface between a source of data and the end user of the data. The end user of the data must be identified to include the requirements for the data and the source supplying the data must also be identified. Data routing is transparent to logical interface entities. Routing of the data should not be a concern to the source and end user because the routing (i.e., direct requirements) is transparent to these entities.

Management subsystem is a data processing subsystem which may interface to a human to determine options and select alternatives for implementation. A management subsystem which has no human interface may support one which does have a human interface, or it may be an artificial intelligence capability which replaces a human, perhaps in unmanned missions.

Mission Critical Function is any function which, if it fails, results in an incomplete mission, a mission abort or a loss of payload.

Mission Ready mode is a system condition wherein all system elements, including hardware, software, human and procedural, are available to enable the system to perform its intended function and the current mission for which it is intended.

Mode is a predefined set of hardware and software configurations, and associated procedures used to organize and manage the conditions of operation for an avionics system's behavior, as planned, pre-planned or directed by a human.

Modular Architecture is an architecture composed of discrete components such that the design of one component depends only on the interface to other components, not on their internal design. A modular architecture is decomposable, understandable, protected, has continuity and is organized in a robust structure. It is desirable that a change in one component has minimal impact on other components. (Adapted from [SSP 30235]).

Network Services Manager (NSM) is a control subsystem which manages peer-to-peer communication between applications software running on distributed processing elements communicating over a network.

Object is something perceptible to the sense of vision or touch or to the mind.

Open Forum is defined as the review of a subject in a public consensus process.

Open Interface Standards are standards that are complete, consistent and published. Open interface standards must be maintained and accepted by a publicly accessible review body.

Open Specifications are public specifications that are maintained by an open, public consensus process to accommodate new technologies over time and that are consistent with international standards. The public consensus process for open specifications must be maintained and accepted by an open forum. (POSIX P1003.0 Draft 14 Guide)

Open System is a system that implements sufficient open specifications for interfaces, services, and supporting formats to enable properly engineered applications software (POSIX P1003.0 Draft 14 Guide):

- to be ported with minimal changes across a wide range of systems
- to interoperate with other applications on local and remote systems
- to interact with users in a style that facilitates user portability

Open System Interface Standards are standards that provide for open specifications of open systems.



An Open System Application Program Interface is a combination of standards-based interfaces specifying a complete interface between an application program and the underlying application platform and is divided into the following parts (POSIX P1003.0 Draft 14 Guide):

- Human/Computer Interaction Services API
- Information Services API
- Communication Services API
- System Services API

Open Systems Architecture is defined as an architecture for an open system using open specifications. It consists of a structure of interconnected functional subsystems (i.e., black boxes) using non-proprietary communications, based on open specifications for interfaces, and providing high level standard services. The interface between the application software and the underlying application platform must be based on an Open System Application Program Interface. To be open, the architecture must be extensible through the addition of subsystems, services and resources following open specification rules.

Open System Environment (OSE) is the comprehensive set of interfaces, services and supporting formats, plus user aspects for interoperability or for portability of applications, data, or people, as specified by information technology standards and profiles. (POSIX P1003.0 Draft 14 Guide)

Operating System (OS) is the layer of software that isolates services and application software from the application platform hardware element. The OS provides services for at least management, allocation, and deallocation of the processor, memory, timing and input/output (I/O) processing resources for application and service software.

Operationally Ready mode is a system condition wherein most system hardware, software, human and procedural elements are functioning correctly, but not all subsystems are configured as needed for a mission to be performed.

Portability is the ease with which software can be transferred from one platform, application or information system to another. (POSIX P1003.0 Draft 14 Guide)

Profiling is the process of selecting a set of one or more base standards, and where applicable, the identification of chosen classes, subsets, options, and parameters of those base standards, necessary for accomplishing a particular function. (The profile selection process is discussed in section 6 of [POSIX91]).

Protection is defined to mean that the architecture will limit the effect of abnormal conditions in design elements at run-time to just the affected modules or as a minimum will limit the propagation of abnormal conditions.

Red-tagged mode is a system condition wherein sufficient system hardware, software, human or procedural elements are failed that the system cannot operate at all.

Requirements Architecture is an architecture that can be tailored for design implementation based on actual system requirements.

Robustness is the measure of a system's ability to support continued functioning under abnormal operating conditions.

Safety Critical Function is any function which has an associated condition, event, operation, process, equipment or system (including software) with the potential for major injury or damage, adapted from [SSP 30235].

Service Subsystem is service software on an applications platform, which provides transparent services to the using control or data processing subsystem.

Service functions are usually widely replicated in support of many control or data processing subsystems. This wide replication of functionality is a key determining characteristic in defining an individual process as a service in this methodology. Services are critical to system operation, not to mission or vehicle operation per se. An example of a service function is a Report Generator since many applications and control subsystems must generate reports; here, they call on the report generator service which knows how to look up the table defining the applications/control report, how to format the format for completion, how to find the data to fill the report fields with, and how to route the report for distribution based on a predefined distribution list. High level standard services are services such as timing, distributed data handling and fault tolerance, which may have different needs when viewed as a multi-processing system than when considered as a single processor system.

Source is the originator of data passed across a logical interface.

Space Data System - see Data System.

Space Data System Services - Data System Services.

Space Generic Open Avionics Architecture (SGOAA) is defined as the target open architecture standard being developed to provide an umbrella set of requirements for applying a generic architecture interface model to the design of specific avionics

hardware/software systems. This standard defines a generic set of system interface points and establishes the requirements for applying appropriate low level detailed implementation standards to those interfaces points. The generic core avionics system and processing hardware architecture models provided by the standard are robustly tailorable to specific system applications and provide a platform upon which the generic interface model is to be applied.

Standard Data Services Manager is the interface handling subsystem that manages the services that process requests for interaction between sensors, effectors, applications software and other services.

Standard is a document established by consensus and approved by a recognized body, that provides, for common and repeated use, rules, guidelines, or characteristics for activities or their results, aimed at the achievement of the maximum degree of order in a given context.

Standardized Profile is a balloted formal, harmonized document that specifies a profile. (POSIX P1003.0 Draft 14 Guide)

System is defined as the composite of equipment, material, computer software, personnel, facilities and information/procedural data that satisfies a user need. (Electronic Industries Association Bulletin SYSB-1)

System Hardware Architecture is an architecture consisting of the set of hardware resources in a configuration of distributed computers, memories, buses and network elements.

Some of the characteristics that determine the nature and requirements for a system hardware architecture are the number of processors, their type and topology, the speed and size of shared memory available, the local memory of each, the bandwidth and access to communications media, and the interfaces available for use by people, applications and platform software services in the hardware.

System Software Architecture is an architecture consisting of the elements and interfaces between software components in a system.

System Services Software is common software, independent of application software, which is needed to run applications software and enable it to interface to data within a system or across the EEL. This is similar to the POSIX entity, system software, which is defined as the application independent software that supports the running of application software.

Task is defined as a software entity that is executed in parallel with other parts of a software program to perform an action. [BOOCH87]

Understandability is defined to mean all requirements related to a subject can be found and viewed together, and individually and jointly understood by the analysts and designers.

A Utility Function is a function which if it fails will result in no control loss or unsafe condition.

**APPENDIX B**  
**ARCHITECTURE SPECIFICATION TABLES**



## **B. ARCHITECTURE SPECIFICATION TABLES**

### **B.1 INTRODUCTION**

The data presented in this appendix provide examples of potentially selectable specific performance parameters, and are excerpts from [BOE91]. Satisfying these performance parameters, such as less than one second for real-time transport delay, is a design implementation requirement dependent upon the needs of a specific system and as such are outside the scope of the SGOAA.

## **B.2 MISSION AND VEHICLE REQUIREMENTS**

[BOE91] was prepared to provide avionics flight data systems (FDS) requirements as shown in Tables B-1, B-2 and B-3. They are applied to the following four classes of target vehicles:

- Class 1 is made of large manned vehicles consisting of independent segments with flight critical real-time needs and with life support, such as the Trans-Earth Injection Stage (TEIS - Piloted), the Trans-Mars Injection Stage (TMIS - Piloted), the SSF/ACRV and Orbital Assembly Nodes.
- Class 2 is made of large manned vehicles consisting of a single segment, reduced real-time flight critical needs and life support, such as the Earth Crew Capture Vehicle (ECCV - Lunar), Earth Crew Capture Vehicle (ECCV - Mars), Lunar Excursion Vehicle - Piloted, Mars Ascent Vehicle, the Mars Descent Vehicle, Lunar Habitat/Lab/Support Modules, Martian Habitat/Lab/Support Modules, NSTS Shuttle Follow-On, Personnel Launch System, Lunar Rover (pressurized), Martian Rover (pressurized), Ballistic Vehicle - Mars, Ballistic Vehicle - Lunar, and Lunar Transfer Vehicle - Piloted.
- Class 3 is made of smaller manned vehicles consisting of a single segment, reduced criticality requirements, possibly with no life support, such as EVA/EMU, Manned Maneuvering Unit - Phobos Type, Lunar Rover (unpressurized), Mars Rover (unpressurized), and Modular Space Sub.
- Class 4 is made of unmanned space flight or planetary surface vehicles or facilities, such as the Aero Assist Flight (AFE) experiment, Communications Orbiter, Earth Orbit Vehicles, Lunar Excursion Vehicles, Mining and Manufacturing Equipment for Lunar or Mars, Mars Sample and Return, Meteorological Stations, Observatory Equipment for Lunar or Mars, Robotic Rovers for Lunar or Mars, Science Equipment for Lunar or Mars, Site Recon Orbiter, Trans Mars Injection Stage - Cargo, and Lunar Transfer Vehicle - Cargo.



Table B-1. SATWG Avionics Architecture Vehicle Study Class Definition

	A	B	C	D	E	F	G	H	I	J
	NASA/JSC				SATWG Avionics Architecture	Vehicle Study Class	Definition			
1				Boeing					9/10/91	
2				R. A. Flanagan (200)773-8613						
3					Class 1 (Manned, multi-segment)	Class 2 (Manned, single segment)	Class 3 (Manned, EVA)	Class 4A (Unmanned, multi-segment)	Class 4B (Unmanned, single segment)	Class 4C (Unmanned, EVA)
4	Flight Hardware Element	CLASS								Remarks
5	AFE	4								no study data
6	Communications Orbiter	4								
7	ECOV, lunar	2			yes					
8	ECOV, Mars	2			yes					
9	ETO	4								
10	EVA/EMU	3				yes				
11	Lunar Hab/Lab/support equipment	2			Initial Outpost/Hab					
12	Martian Hab/Lab/support equipment	2			yes					
13	LEV-C	4					yes, common			common w/LEV-P
14	LEV-P	2			yes, if PUV male					common w/TV-P
15	LTV-G	4								
16	LTV-P	2			yes					
17	MAV	2			yes					
18	MCL	4						yes, not studied		
19	MDV	2			yes					
20	Mining and Manufacturing Equipment, lunar	4						yes, not studied		
21	Mining and Manufacturing Equipment, Mars	4						yes, not studied		
22	MMU, Phobos type	3				yes				
23	Mars Sample & Return	4								
24	Meteorological stations	4			yes					
25	NETS Shuttle follow-on	2								
26	Observatory Equipment, lunar	4								
27	Observatory Equipment, Mars	4								
28	Orbital Assembly Node	1			yes, if SSF type					
29	PLB	2			yes					launch vehicle
30	Robotic Rover, lunar	4							yes	
31	Robotic Rover, Mars	4							yes	
32	Rever-presurized, lunar (PUV)	2			yes					
33	Rever-presurized, Martian	3				yes				
34	Rever-presurized, Martian (PUV)	2			yes					
35	Rever-presurized, lunar	3				yes				
36	Science Equipment, lunar	4							yes, not studied	
37	Science Equipment, Mars	4							yes, not studied	
38	Site Recon Orbiter	4						yes, not studied		
39	SSF/ACRV	1 and 2			yes, Hab/Lab/EMESA	ACRV				
40	TEIB	1			yes					common w/MPV
41	TMIS (MCV)	4					yes, common			
42	TMIS (MPV)	1			yes					
43										
44										
45	Class 1 is large spacecraft manned vehicle made of independent space segments with flight critical real-time requirements and with life support									
46	Class 2 is large manned vehicle (spacecraft or planetary surface) with reduced flight critical real-time data requirements and with life support									
47	Class 3 is smaller manned vehicle with lower critical requirements than classes 1 or 2 and no life support									
48	Class 4 is unmanned spacecraft flight vehicle, planetary surface vehicle or planetary surface facility									
49										

Table B-2. SATWG FDS Architecture Mission Set Requirements

	A	B	C	D	E	F	G	H	I	J	K
52											
1	NASA/JSC	Boeing									
2		R. A. Flanagan	(206) 773-9513					updated	9/11/91	Preliminary - not coordinated	
3											
4	Flight Element	Case Type (3)	Phase A (2)	IOC (4)	Orbit Insertion	In-space Assembly	Cargo Transfer		Cargo Delivery	Life Support	Rendezvous
5	Lunar Missions										
6	Reconn Rover (manned) (8)	(8TV) LTV-P		2004	LLO	n/a	Dual role with LTV		n/a	yes (1)	
7	Manned/Robotic Rover	LEVPU (9)		2002	n/a	LTV Cargo			n/a	no	
8	LTV Program Transfer Vehicle	(8TV) LTV-P	1993	2004	LLO/LEO	yes	node to LLO 45 m		crew of 8 and 15 m	yes	with return node?
9	LTV Program Excursion Vehicle	(8TV) LEV-P	1993	2004	LLO	yes	LLO to surface 20 m		crew of 8	lander/Hab	with LTV
10	LTV Program Earth Entry Vehicle	(8TV) ECCV		2004		LTV Cargo	n/a		crew of 6	yes	
11	Lunar Hab/Lab	Planetary Surface	1995/1997	2004	n/a	LTV Cargo	n/a		crew of 5	SIOC/12-18OC	
12	Rover: unpressurized	Planetary Surface	1994	2004	n/a	LTV Cargo	500kg		2 to 4	no	with Lunar Hab
13	Rover: pressurized	Planetary Surface	1994	2004	n/a	LTV Cargo			4		100km
14	Man/Robot/Assted Missions										
15	Man/Robot/Assted Missions										
16	Site Recon Orbiter			1998	LMO	n/a	n/a		n/a	no	
17	Reconn Orbiter (manned)	(8TV) MPV		2014	LMO	n/a	n/a		n/a	yes	
18	Communications Orbiter (TNM) (1)			2007	n/a	n/a	n/a		n/a	no	
19	Man Sample & Return			1998	LMO/LEO?	n/a	return sample		sample return	no	with orbiter
20	Meteorological stations (TNM)	Planetary Surface		2014	n/a	MTV Cargo	n/a		n/a	no	
21	MTV Program Transfer Vehicle	(8TV) TMIS/TEIS/MPV	1996	2014 (7)	LMO/PLEO	yes	to LMO		crew of k + 100 m	crew & simulation	with return node?
22	MTV Program Excursion Vehicle	(8TV) MOV/MAV	1998	2014	LMO/PLEO	yes			crew of k + 1	yes, crew of 5	with MTV
23	MTV Program Earth Entry Vehicle	(8TV) MPV/ECCV	1993	2014	n/a	MTV Cargo	n/a		crew of k + 1	yes, crew of 6	
24	Habit/Lab	Planetary Surface	1993	2014	n/a	MTV Cargo	n/a		crew of 2 to 4	yes, up to 18	n/a
25	Probe-type MMU	EVA		2014		MTV Cargo			crew of 2 to 4	no	6 km
26	Local rover: unpress	MEVPU (9)		2014	n/a	MTV Cargo	500 kg		2 to 4	no	with Meridian Hab
27	Science Rover: unpress	Planetary Surface	1996	2014	n/a	MTV Cargo			2 to 4	no	with Meridian Hab
28	Outpost Construction Rover: pressurized	Planetary Surface		2014	n/a	MTV Cargo	1000 kg		4	yes	1000 km
29	Other Manned Support Vehicles										
30											
31	Adv NSTS Shuttle	ETO		?	LEO	n/a			> 400lb + crew	yes	with 88F
32	PL 8	ETO		?	LEO	n/a			crew of 8 to LEO	yes	with 88F
33	Aerocast Flight Experiment (AFE)			?	yes	n/a	n/a		n/a	no	n/a
34	SBFACRV	Orbital Node System		1999 PMC	reboost	yes			n/a	yes	
35	LTV Assembly Node	Orbital Node System		?	LEO		HLLV cargo to node		n/a	yes	
36	MTV Assembly Node (2)	Orbital Node System	1994	IOC-1v	node 2	yes	HLLV cargo to node		n/a	yes	LLO (2)
37											
38											
39	note 1 See OEXP Vol 1 4.3.2.1 Partitioning of TNM and Related Mission Support Functions										
40	note 2 OEXP Vol 1 references order as LLO, L2 and L1 based on mass performance										
41	note 3 Nomenclature from OEXP FY99 Vol 1, Section 3										
42	note 4 Stafford Group references; generalized Initial Operational Capability dates for the four architecture										
43	note 5 Abort principles require both abort to Surface/Orbit and options to the Commander										
44	note 6 Enroute phases, Ascent/Descent phases and planet surface operations require										
45	multiple levels of parallel redundancy with high reliability and low maintenance.										
46	See Stafford, Considerations and Constraints section for Crew Safety, page 18										
47	note 7 Aggressive Mars option 2008, Stafford, p59										
48	note 8 SEI STS Interim Rpt										
49	note 9 90-day Study Planetary Surface Operations										
50											

Table B-2. SATWG FDS Architecture Mission Set Requirements  
(continued)

L	M	N	O	P
52	Mission Set- Continued			
1				
2				
3				
4	Dock/Transfer	Orbital Observation	Descend (S)	Ascend (S)
5				Surface Ops
6	passive element	plat ops/obs/telepresence		n/a
7				
8	passive element with LTV	dual role	n/a	n/a
9			to lunar surface	
10	with LTV	n/a	to earth surface	n/a
11	passive	n/a	n/a	45-180 days
12	n/a	n/a	n/a	Apollo type
13	with Hab/Lab	n/a	n/a	construction
14				
15				
16	n/a	remote sensing	n/a	n/a
17		plat ops/obs/telepresence	n/a	n/a
18	n/a		n/a	n/a
19	with orbiter	?	to planet surface	to LMO
20	n/a	n/a	n/a	acquire sample
21	passive element	plat ops/obs/telepresence	n/a	TNIM (1)
22	with MPV		to planet surface	n/a
23	with MPV	n/a	to earth surface	n/a
24	passive element	n/a	n/a	30-100,100-800d
25	Phobos		yes	yes
26	n/a		n/a	TNIM (1)
27			n/a	TNIM (1)
28	with Hab/Lab		n/a	construction
29				
30				
31	with SSF	extended ops/science	yes	to LEO/SSF
32	with SSF	n/a	yes	to LEO/SSF
33	passive		yes	yes
34	ACRV/PLS/Shuttle	science	ACRV	n/a
35	manifest elements	n/a	n/a	n/a
36	manifest elements	n/a	n/a	n/a
37				
38				
39				
40	yelem			
41	Capability			
42	hicle			
43	ole			
44	n Stage			
45	n stage			
46	to Mgt			
47				
48				
49				
50				

Table B-3. SATWG FDS Architecture Service And Interface Requirements

1	A	B	C	D	E	F	G	H	I	J	K
	NASA/JSC	Boeing									
2		R. A. Flanagan (208) 773-8813							updated	9/18/91	
3											
4											
5			Interface operator	Interface sensors	Interface memory also	Interface latency/fitter	Interface Signal	Interface interfunction	Interface topology	Interface effects	Interface external
6				type/number	hw/dw		Conditioning	communication		number/type	
7											
8											
9	4.2 Interface Requirements										
10	4.2.1 user class										
11	4.2.1.1 human										
12	4.2.1.1.1 on-board crew member										
13	Controls-Displays		cmd input	kybd, atk, sw, voice				from all		16 cameras	
14	Warning, Caution, Advisory		alarm/audio/data	voice syn/light				to veh sys mgt/mon		25 lights/segment	
15	Training & Simulation		ops & gen education							20	
16	Medical Support Systems									16	
17	4.2.1.1.2 off-board remote access										
18	ground/node controllers										
19	experimental scientists			payload							
20	4.2.1.2 External Interface										
21	Vehicle System Mgt Support										
22	Operations Sequencing-Mgt		sim/training ops		3c:22 GB						
23	Systems Monitoring				3c:186 MB	1: class 2				20/segment	
24	Health Management				3c:1115 KB	1: class 2				10sw/segment	
25	Health Monitoring					1: class 5					
26	GN&C Support			300 to 3000 (dumb)		1: class 5		from all			gateway/CAT
27	Navigation							to Health Mon			gateway/CAT
28	Guidance			4c:50		1: class 1		to Guidance/Payload			
29	Mission Sensor Management			none		1: class 2		to flight control			
30	Rendezvous			Imaging/science				to guidance			
31	Docking		sim/training ops	10 radar/laser		1: class 2		to guidance		beam steering	
32	Flight Control		sim/training ops	10 radar/laser		1: class 2		to guidance			
33	Attitude Determ-Control			9 air data/alt		1: class 2		to attitude determin & Control			
34	Ejector control Support							to propulsion			
35	Propulsion Support							to Health Mon		TVC/manipulator	
36	propellant Mgt							to Health Mon		valves	
37	Main Propulsion Control			20/tank		1: class 2				valves	
38	Reactor Control		sim/training ops	50 engine		1: class 1				32 valves/engine	
39	Auxiliary Propulsion			64		1: class 1				60	
40	Environment Support					1: class 2				32 valves	
41	ECLSS		data/audio	100		1: class 5		to Health Mon		100 valves	
42	Thermal Control			60		1: class 5				valves/sw	
43	Electrical Pwr Control Support							to Health Mon		25 valves/sw	
44	Power Source Control			35						sw	
45	Power Distribution Mgt			10/segment				to Health Mon			
46	Communications Support										
47	Communication Links		video/audio/data	antenna sw and pointing						8 antennas	2-video 10 Mbps
48	Tracking			antenna sw and pointing						transponder/recv/hmler	
49	Telemetry Formatting										
50	Payload Services Support							to Health Mon			2:50 Kbps
51	Deployment										
52	Data & power interface		sim/training ops							mech+sttuc actuators	
53	Range Safety Support		n/a	n/a	n/a	n/a	n/a	n/a	n/a	gateway/bridge	gateway/CAT
54										n/a	n/a

Table B-3. SATWG FDS Architecture Service And Interface Requirements  
(continued)

L	N	O	P	Q	R	S	T	U	V	W	X
1											
2											
3											
4											
5	Interface	service	service	service	service	service	service	service	service	service	service
6	time gen/dst	security	System Control	A1	processing type	throughput	logic	topology	data storage and retrieval	criticality	interrupt handling
7							sequencing				
8											
9											
10											
11									3:		
12									40 MB	fl/mission	
13	user	privacy/security			numeric	very high	high		2:200 MB	safety	1:2:
14					symbolic/numeric	low			2:200 MB	mission	1:2:
15	user	privacy/security		symbolic	symbolic/numeric	very high	high		2:200 MB		
16				expert	symbolic/numeric						
17									4:remote	fl/mission	
18		privacy/security							4:remote	mission	
19		privacy/security									
20		privacy/security			symbolic/numeric	180 MIPS					
21											
22	control	privacy/security	1:2:autonomous		numeric		high	250 meters	0.2 MB	fl/mission	1:2:
23			3-6:autonomous		numeric			determ/central	5 MB	flight	
24		privacy/security	3-6:autonomous	blackboard	symbolic/numeric	very high	high	central		mission	
25	user		3-6:autonomous	blackboard	numeric				2:200 MB	fl/mission/safety	
26											
27	source		3-6:autonomous		2:40 MFLOPS	high	high	determ/central	5 MB	fl/mission	1:2:
28	user		3-6:autonomous		symbolic/numeric	very high	high	determ/central	5MB	fl/mission	1:2:
29			3-6:autonomous		symbolic/numeric	high	high		1 MB	mission	
30			3-6:autonomous		symbolic/numeric	high	high	determ/central	3 MB	fl/mission	1:2:
31	grade 1				numeric	high		determ/central	3 MB	fl/mission	1:2:
32	grade 1		3-6:autonomous		numeric	high	high	determ/central	5 MB	flight	1:2:
33	grade 1		3-6:autonomous		numeric	high	high	determ/central	5 MB	flight	1:2:
34					numeric	high			0.1 MB		
35											
36	grade 1		3-6:autonomous		numeric	high	high	determ/central	0.1 MB	safety	1:2:
37	grade 1		3-6:autonomous		numeric	high	high	determ/central	1 MB	flight/safety	1:2:
38	grade 1		3-6:autonomous		numeric	high	high	determ/central	1 MB	flight/safety	1:2:
39	grade 1		3-6:autonomous		numeric	high			1 MB	flight	1:2:
40									10 MB	safety	
41	grade 2		3-6:autonomous		numeric	low			0.2 MB	safety	
42	grade 2	45 kW	3-6:autonomous		numeric	low					
43											
44	grade 1				numeric	low	high	determ/central	1 MB	fl/mission/safety	1:2:
45	grade 2				numeric	low		determ/central	1 MB	fl/mission/safety	1:2:
46									5:0.2 MB	mission	
47	source update	1:2:up/down			numeric	med		determ/central	22 GB	mission	1:2:
48					numeric	med				mission	
49	user				symbolic/numeric	high		determ/central		mission	1:2:
50											
51					numeric	med	high	determ/central		mission	
52	user	privacy/security			symbolic/numeric	med		determ/central		mission	
53	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
54											

### B.3 PERFORMANCE PARAMETER REQUIREMENTS

A potential set of FDS standard sensor interfaces are shown in Table B-4. FC1 is defined as Category Flight Critical (FC) Level 1. Categories and levels are defined in Table 2-1.

Table B-4. Sensor Interface

SENSOR INTERFACE	NO. REAL-TIME INTERFACES FC1A, MC2A, SC3A	NO. NON-REAL-TIME INTERFACES MC2B, SC3B, NC4B
Data bus compatible type	0 to 200	n/a
Data bus non-compatible type	n/a	100 to 24,000
High data rate type	0 to 10	0 to 10

A potential set of FDS standard effector interfaces are shown in Table B-5.

Table B-5. Effector Interfaces

EFFECTOR TYPE	NO. FLIGHT CRITICAL INTERFACES FC1A	NO. MISSION CRITICAL AND SAFETY CRITICAL INTERFACES MC2A, SC3A MC2B, SC3B	NO. NON- CRITICAL INTERFACES NC4B
Data bus compatible type	TBD	TBD	TBD
Data bus non-compatible type	TBD	TBD	TBD
total	50 to 500	25 to 1,000	25 to 100

The Bit Error Rates potentially applicable shall be as defined in Table B-6, among onboard users and between users and Communications and Tracking C&T.

Table B-6. Bit Error Rate Characteristics

	GRADE		
CHARACTERISTIC	I	II	III
Bit Error Rate (BER)	10E-12	10E-8	10E-5
Header BER	10E-12	10E-12	10E-12

A potential set of FDS message handling priorities shall be as shown in Table B-7.

Table B-7. Priority Message Transfer Latency (Maximum Time)

MESSAGE PRIORITY	FC1	MC2	SC3	NC4
Background	TBD	TBD	TBD	TBD
Normal Periodic	TBD	TBD	TBD	TBD
Normal Event	TBD	TBD	TBD	TBD
Expedited	TBD	TBD	TBD	TBD
Emergency	TBD	TBD	TBD	TBD

The FDS shall provide for implementing throughput within the range of values listed in Tables B-8 and B-9 for normal operation of applications available to service requester. Values shown assume no interrupts or other network dependent overhead.

Table B-8. Processing Throughput Capacity

THROUGHPUT PROCESSING	MIPS	
	MINIMUM	MAXIMUM
Critical, Real-time	0.3	627
Critical, Non-real-time	0.5	160
Non-critical	1	100

Table B-9. Network Throughput Capacity

INTERFUNCTION NETWORK THROUGHPUT	MBPS	
	MINIMUM	MAXIMUM
Critical, Real-time local network	1	10
Critical, Real-time internetwork	1	10
Critical, Non-real-time local network	0.5	10
Non-critical local network	1	10
Non-critical internetwork	10	150

The FDS shall segregate critical data (FC1A, MC2A, MC2B, SC3A, SC3B) from noncritical data (NC4B) and provide data protection. Potentially allocated FDS memory space available

to software applications are listed in Table B-10. Application space addresses may be memory load dependent.

Table B-10. Allocated Memory Space

ALLOCATED MEMORY SPACE	MEGABYTES	
	MINIMUM	MAXIMUM
Primary Volatile Memory - Critical	1	350
Primary Volatile Memory - Non-critical	TBD	TBD
Primary Non-volatile Memory - Critical	0.1	360
Primary Non-volatile - Non-critical	TBD	TBD
Secondary Memory, On-line - Critical	100	2120
Secondary Memory, Off-line	TBD	TBD
Secondary Memory, Communications Buffer	0	1500

The FDS shall generate and distribute timing within requirements listed in Table B-11.

Table B-11. Timing Requirements

TIME	RESOLUTION (MILLISECOND)	ACCURACY (MILLISECOND)
Local Network:		
Time Reference	1	TBD
Event Timing	1	TBD
System Time (e.g., GMT)	1	TBD
Mission Elapsed Time (MET)	1	TBD
Internetwork (global):		
Time Reference	1	TBD
Event Timing	1	TBD
System Time (e.g., GMT)	1	TBD
Mission Elapsed Time (MET)	1	TBD



**APPENDIX C**  
**ARCHITECTURE REVIEW COMMENTS AND RESPONSES**



## **APPENDIX C**

### **ARCHITECTURE REVIEW COMMENTS AND RESPONSES**

#### **C.1 EAGLE ENGINEERING, INC. COMMENTS**

Comments were provided by Eagle Engineering, 16915, El Camino Real, Suite 200, Houston, Texas 77058, (713) 283-6000, dated May 27, 1992 are provided herein. These comments are on the Preliminary Draft of the "Space Generic Open Avionics Architecture (SGOAA) Standard", dated 2 April 1992.

This activity is an appropriate one for SATWG to sponsor, and final adoption of the product as a standard should provide excellent return on the investment in its preparation.

##### **C.1.1 GENERAL COMMENTS**

###### **General Comment 1)**

The difficulties of dealing with an open Avionics Architecture when most onboard systems can (with the exception of the organizations representing them) be classified as "Avionics" are very real and apparent in this document. It will be some time before system engineering organizations evolve which can adequately get to grips with defining an overall vehicle architecture and to be able to take advantage of a standardized architecture--however, this document should be pursued vigorously as one means of providing the necessary education.

###### **Response:**

We agree, one purpose of this document is to provide sufficient technical description and rationale to serve as a strawman for coming to grips with the necessary avionics definitions and architecture for all space avionics and an overall vehicle architecture. Use of standardized architectural elements appears essential in the current funding constrained environment.

###### **General Comment 2)**

The term "function" as used in Table 2.1 "Critical Function Categories" needs to be rigorously defined in this document. The SSF program has shown the futility of attempting to define redundancy, reliability, fault tolerance, etc. requirements based on traditional subsystem requirements. This proposed SGOAA standard clearly recognizes this, but stops short of providing a clear definition of the word. With the

continuing blurring of the distinctions between traditional Data Systems, Avionic Systems and Non-avionics systems the term "function" (or some other term meaning something like "a complete path from command initiation to physical result") becomes increasingly important in the interpretation of requirements and must be clearly understood by all involved in the design process. An example might be the transport delays and timing constraints given in the document. Do these apply to the total function or to the.- Avionics interface as defined in Fig. 2.3?

**Response:**

Agree. Function has been defined in Appendix A. In addition, parametric values such as transport delays and timing constants have been removed from the requirements section. These parameters are design requirements to be based on the needs of a specific system and as such are outside the scope of a generic architecture such as the SGOAA.

**General Comment 3)**

At this stage in the development of a document such as the proposed standard, it is often desirable to provide information of an explanatory nature which would not necessarily find its way into the completed version. This early draft could benefit by additional text of this type in several areas. It is suggested that where the information is obviously tutorial in nature, that the discussion is so identified, e.g. in italics or different font, etc.

**Response:**

Agree that explanatory and rationale type material must be provided. This document is equivalent to the .0 document of POSIX, in that it attempts to provide a full technical description of the architecture. This document is not the standard per se to be proposed; that will be provided in a separately released document with just the specification data needed and appropriate to a standard. An executive summary and briefing materials will also be published.

**General Comment 4)**

It is suggested that verifiability be considered in this Standard. Difficulties and costs associated with verification have become increasingly more evident in the history of NASA programs and, indeed, verification is currently a major issue in the Space Station Freedom Program. Consideration might be given, for example, to promoting the partitioning of flight software into simply "critical" vs. "non-critical" parts (or

perhaps into the categories of Table 2-1) for the purpose of verification. Emphasis would then be put on the preflight verification of the more critical parts, with possibly the exclusion totally of verification of parts with little or no criticality.

Another application might be the consideration of verification in deciding the command path architecture discussed in Section 3.4.1.1. Although from one perspective multiple command paths provide "robustness", they also impose a verification penalty which may be intolerable. In many cases it will prove more cost effective simply to provide redundant strings of single command path hardware.

**Response:**

Verifiability is an important facet of using a standard. Whether and how to include it here is an open question.

**General Comment 5)**

The document provided for review contained computer generated diagrams which were totally illegible. This should be corrected in subsequent versions.

**Response:**

The final version of this document will be submitted to a formal documentation publication process with no poor quality diagrams at all. In the interest of speed and cost minimization, the drafts are published more expediently. Every attempt will be made to provide quality diagrams, however, multiple stages of reproduction cause sufficient degradation of quality after the distribution is made to first delivery sites that blurry diagrams may nevertheless occur.

**C.1.2 DETAILED COMMENTS**

**Paragraph 1.0**

The term "open architecture" has become fairly standard usage in the SATWG community, but for others it might be helpful to provide a definition here in the Introduction.

**Response:**

Agree. "Open Architecture" is defined in Appendix A.

**Paragraph 2.1**

Fig. 2.3 defines the avionics boundary as excluding the sensors/actuators, etc. This is a reasonable approach at this stage but might be better accepted if the title was more

limited, i.e., "Avionic Core" or "Avionics Data, Command & Control Architecture". Also, along the lines of General Comment No. 1, it might be worth pointing out in the discussion that from the point of view of defining an overall vehicle architecture there is no essential difference between how the Environmental Control System (ECS) and a traditional Avionics System such as GN&C should be handled.

Another bubble which might be added in the outer area of Fig. 2.3 is "Instrumentation". Although most systems have instrumentation to some degree, most do not have enough for what has been called "development flight test" in the past, and other, non-electronic systems such as structures must usually have instrumentation added for any meaningful in-flight monitoring.

**Response:**

Use of the term avionics core architecture is an excellent idea. Determination of the differences in handling each avionics end-user application function (such as ECS or GN&C) requires architectural analysis. The results of such analysis would then be reflected in the overall avionics architecture. Note that the Architecture Interface Model treats the interfaces between the hardware and software in each applications subsystem the same. For the purposes of clarity, each key subsystem in the avionics should be explicitly identified in the avionics architecture diagram to avoid neglect or the appearance of neglect.

Agree, instrumentation such as development flight instrumentation (DFI) and other key interfaces need to be and are being addressed. The differences (in any) between developmental flight avionics and production flight avionics also need to be considered.

**Paragraph 2.1.1.3.2 (third bullet)**

It is not clear why the assumption is made that sensors and effectors have embedded firmware. Although many do have terminations compatible with data busses such as MIL-STD-1553, some do not and the latter may be more cost effective for simple applications. (This is not to promote the proliferation of MDM's with A/D and D/A conversion capability, but the option to accommodate them might be advisable in a standard such as this.)

**Response:**

The intention was to describe the most general (i.e., the "shopping list" approach) case of sensors and effectors some using firmware and some not, with the option

provided to use an MDM to perform some signal consolidation and pre-processing, without specifying whether the specific low level processes such as analog-to-digital conversion are performed in one or the other. This will be clarified in the next release.

**Paragraph 2.1.1.3.2 (fourth bullet)**

There is another perspective in the realm of "intermediate level processing" which might be considered. The position might be taken that the time has come to do away with "intermediate processing" altogether, "smart MDM's" included, considering the increasing compactness of flight computers while simultaneously providing ever increasing processing capability. Such an approach would depend on input/output/conversion processing in the "back section" or background of the main computer (SDP), and the primary advantage would be a reduction in the number and types of different onboard computers. The main disadvantages would be a more difficult job of integrating all software in the main computer, and a de-emphasis on an architecture which is "backward compatible" with existing hardware, primarily sensors and effectors.

**Response:**

There is an obvious computing trend toward distributing computational power out to the users and end devices needing processing, rather than to continue relying on centralized computing. This avionics core architecture takes no position on this trend. The core architecture allows use of either centralized, distributed or some hybrid by providing all the key elements to support either, in line with the idea that the architecture is a "shopping list" of all (or at least most) potentially needed elements which will have interfaces which have been apriori verified to be compatible or interoperable. Note that a centralized computing scheme is just a special case of a distributed system with just one computer providing all required functions instead of multiple computers providing the required functions in the distributed case.. Use of a specific distribution scheme must be determined on a case-by-case basis depending on the needs of the specific mission being addressed.

**Paragraph 2.1.1.3.3 (sixth bullet)**

It would be helpful if the acronyms used in Figure 2.5 were used in this paragraph. (The terms "SDP" and "MDP" in the text are not used in the figure, and "EP" is used for "effector processing" in the text but used for "embedded processor" in the figure.)

**Response:**

Agree, consistency will be improved in the next draft. Some inconsistency was allowed to pass in the preliminary draft to avoid delays in distributing the document at the April 1992 SATWG meeting.

**Paragraph 2.1.2**

The Reference Model Definitions section should be deleted in favor of a section containing definitions for the complete document. It should be closely scrutinized for omissions. "Systems Management" and "Function" are two essentials. Although the definition of the "Avionics System" appears to come from a reputable source, it does contain several inconsistencies along the lines of General Comment 1) and efforts should be made to update this definition as acceptance is obtained.

**Response:**

Agree. The next draft will accumulate all definitions for the entire document in Appendix A. Function will be added to the list of definitions. Systems Management is not used in the SGOAA report, except as part of the phrase Data System Management, which is defined in the report. Specific suggestions or recommendations for specific definitions are welcomed to improve the robustness of the terminology and semantics used.

**Paragraph 2.2.2**

Table 2.1 Critical Function Categories is introduced into the requirements with no explanation as to source or background and appears to contain several arbitrary requirements. Because of its overriding effect on Avionics architecture the source should be given along with an adequate discussion of the rationale.

**Response:**

Much of this work was based on an uncompleted study being performed by Boeing Space Data Systems. Parametric values such as transport delays and timing constants have been removed from the requirements section. These parameters are design requirements to be based on the needs of a specific system and as such are outside the scope of a generic architecture such as the SGOAA. In addition, Safety Critical is being changed from a Level 2 criticality to a level 1 criticality. This critical function table now is the same as is presently defined for the Space Station Freedom Program.



**Paragraph 2.2.3.4.2**

Suggest a different heading for this. The present one does not seem to fit the Control Center or Prelaunch facility interfaces.

**Response:**

"Crew " was changed to "Human".

**Paragraph 2.2.3.6 and following**

It would be helpful if the rationale were provided for the values used in Tables 2-2 through 2-13. For example, the values of 627 MIPS in Table 2-8 and 2120 megabytes in Table 2-10 seem large without some explanation. Also, units are missing in some tables.

**Response:**

The rationales are not available. These numbers will be moved to an appendix to separate out supportable material from material which may be correct but which cannot be explained due to changes in supporting staff. See response to General comment 2).

**Paragraph 2.2.4.1.1**

It would seem that if one grade of Message Transfer Service is "Required" another grade could not be "Allowed". Perhaps a better pairing would be "Required" with "Not Allowed" and "Desired" with "Allowed".

**Response:**

Agree, this has been changed.

**Paragraph 2.2.4.1.4**

"Autonomous" should be replaced or defined. Does it mean "Autonomous with respect to the ground", Fully Automatic, or other?

**Response:**

"Autonomous" has been replace with "fully automatic.

**Paragraph 2.2.4.2.4**

Is it the intent of this requirement that "normal operation should be maintained in the presence of two non-simultaneous failures in a flight critical functional path--

these failures being within the interface boundaries of the SGOAA?. If it is, then it should so state

**Response:**

This is clarified in the update. It is now stated that normal operation will be maintained for "failures within the interface boundaries of the SGOAA".

**Paragraph 2.2.4.2.5**

It is believed that a higher percent than 95% for Onboard Fault detection coverage should be achieved during directed Health Monitoring tests. It is believed that the requirement of 99% for Onboard Fault Isolation during normal operation will be difficult and expensive to achieve. Ground rules relating these requirements to the degree of human intervention, (if any), time constraints, percent of onboard resources which can be assigned to the task, etc., must be developed before it is meaningful to include such numbers as this in the standard.

**Response:**

Agree that fault detection is historically in the 99+ percent range for human rated systems, and almost as high for non-human rated systems. This area is under study by several groups, including the SATWG Vehicle Health Management working group, and their results (when available) will be incorporated in future versions of this document. The use of such numbers will be moved to an appendix.

**Paragraph 2.2.4.2.6 through 2.2.4.2.8**

Again, it is suggested that another "ility" worth including is "verifiability".

**Response:**

Verifiability is an important facet of using a standard. Whether and how to include it here is an open question.

**Paragraph 2.2.4.2.8**

The selection of four ORUs seems a little arbitrary. This may be a trifling requirement in some systems and impossible to achieve in others. Consideration should be given to specifying this as a percentage of the on-board ORUs.

**Response:**

Agree, will be changed.

#### **Paragraph 2.3.3.6**

It is suggested that applications software to applications software interfaces, although necessary in some instances, be discouraged because of implications for integrated systems verification.

#### **Response:**

Agree. This has been clarified in paragraphs 2.2.2 and 2.3.3.6.

#### **Paragraph 3.3**

Again, referring back to Figure 2.3, it is recommended that Figure 3-2 have "Instrumentation" as a bubble in the unshaded region.

#### **Response:**

Agree. Instrumentation bubble has been added.

#### **Paragraph 3.4.1.1**

Again, multiple command paths may "provide robustness" but do exact a penalty for verification.

#### **Response:**

Agree, a necessary evil perhaps. This is a program dependent decision. If the particular needs of a specific program do not require these multiple paths, they are not required to be used. These paths are available for the general case. The discussion has been revised to emphasize that there is only one physical path to be verified. The multiple command paths are logical paths which also require verification should a specific program decide to implement these multiple paths..

#### **Paragraph 4.1**

No disagreement was found with the guidelines listed in the conclusions; however, it is believed that the value of applying the SGOAA concept to the assessment of the CLL Data System is overstated. This vehicle has extremely simple requirements and, given the same ground rules, most conceptual designers would have arrived at a configuration similar to the SGOAA or derived one in a shorter time. This is not meant to be a criticism of the SGOAA concept—just that a better example could have been chosen.

**Response:**

Agree that the CLL is a very simple example, however, it is the most-real example tried to date. Also, while simple, it did work, which is very unusual in systems development. The common result of developing a generic architecture and applying it to a real avionics need is that the generic architecture must be so heavily tailored to work at all with adequate performance that there is little left of the "generic-ness" of the architecture.

**Paragraph 4.2**

The recommendations listed appear to be pertinent; however, we believe that one of the more important tasks is the interpretation of NASA requirements (in such areas as Critical Functions; FDIR; Reliability; and Maintainability, for example) so that they may be written unambiguously in a form suitable for a standard. The present document falls far short of this, but unless it is accomplished successfully, the standard will not find ready acceptance. Also, it is recommended that verification be recognized and addressed in the ongoing SGOAA development.

**Response:**

The recommendations list is not complete or validated at this time. Guidelines for the unambiguous writing and interpretation of NASA requirements in general appears beyond the scope of this document. Every attempt will be made to make this document unambiguous by providing clear and full explanations in it. Verification is recognized as an important function which must be addressed. TBD

Eagle Engineering appreciates the opportunity to review this proposed standard, and we hope these comments are beneficial.

## **C.2 MITRE CORPORATION COMMENTS**

Feedback on "Space Generic Open Avionics Architecture (SGOAA) Standard, Proposed SATWG Standard", as reviewed by D.M. Erb, 5/1/92

### **C.2.1 GENERAL COMMENTS**

#### **General Comment 1)**

**Title:** The title might be modified in the light of the SAAP consensus that SAAP ought not to be a "standards-setting" organization.

#### **Response:**

Agree. The title is being changed to reflect that a separate document has been prepared as the proposed standard. It is also recognized that the SATWG is not a standards body and reference to that capacity is being dropped from all titles. The new title of the document will be "Technical Guide to the Proposed Space Generic Open Avionics Architecture Standard".

#### **General Comment 2)**

**Vocabulary:** At this stage of the document, and with multiple authors, inconsistencies are expected. I would recommend that for the next publication, an effort be made to obtain agreement on the use of the terms: reference model, architecture, architecture system architecture, architectural functions, model, configuration, and standard. Then there are times the architecture and an-actual-instantiation-of-the thing-itself-in-space seem to be equated. There is not a clean development of the present document in these regards. NIST has struggled in this area and I have some references that might help should arbitration be required. Definitions, including that of "avionics", would be helpful closer to the front; alternatively, a glossary could be used.

#### **Response:**

An Appendix A containing SGOAA Definitions has been added to the next publication. In addition the document will be cleansed of inconsistencies in term usage.

#### **General Comment 3)**

2.1 I had trouble with this Architecture Background section. I believe it should be crisper in the sense of showing the development of the ideas which support the

current proposals. If there were alternatives that were rejected, perhaps they should be included. A roadmap of issues and decisions here might be the only figure needed. There seems to be a redundancy with later material.

**Response:**

This section has been tightened up. The development of ideas is the point of the whole guide, and would be difficult to summarize at this point. However, min-summaries of ideas will be made throughout the guide at appropriate points. Rather than rejecting alternatives, the development process was one of continual refinement and evolution. The issues and decisions in the development process were documented in [WRA91].

**General Comment 4)**

I have ordered a number of publications on avionics architectures based on work by the Navy and Air Force. I will let you know if they seem to be pertinent. Also, I would recommend that you check the SEI *Generic Avionics Software Specification* document CMU/SEI-90-TR-8, ESD-TR-90-209 by Locke et al, for ideas. Perhaps the work done by Honeywell in the "Future Manned Systems Advanced Avionics Study" published January 1992 should also be referenced.

**Response:**

We are continuing to follow any other architecture-relevant activities, and are interested in understanding the details of the advanced architectures being developed by the U.S. military services, which have been considering this for many years. The referenced documents will be acquired.

**C.2.1 SPECIFIC COMMENTS**

**Paragraph 2.1.1.1**

The second paragraph does not seem to relate to the rational of the first paragraph. I need some help to understand it.

**Response:**

The reference to software is intended to reflect that the functions identified appear to be software elements, so one might be mis-lead into thinking a software architecture is being presented when it is not. This has been re-written to clarify the point.

**Paragraph 2.1.1.2**

Were these key principles intended to be used in the development of the subsequently presented architecture?

**Response:**

Yes, these key principles were used in the development of the architecture.

**Paragraph 2.1.1.3**

"Architecture analysis needs automated tools support." Were automated tools used in the architectural analysis (if any) of the proposed architecture? If so, which ones? What was the output? If not, why is the paragraph there?

**Response:**

Computer aided systems engineering (CASE) tools were used in the development of this architecture, as described in [WRA91]. They consisted of Excelerator/RTS and Cadre Teamwork. The output consists of detailed functional flow and state transition diagrams, of which simplified versions have been presented in this guide.

**Paragraph 2.2.3.4.1**

"crew interface to SGOAA." Surely the crew doesn't need to interface with the architecture?

**Response:**

Agree. What the SGOAA must do is to provide the capability for crew interface to the Avionics system. This paragraph has been modified to reflect this.

**Paragraph 2.2.3.5**

"No application to application interface is allowed." I sensed a conflict with this statement and what I understood from figures 2-2 and 2-9.

**Response:**

The sense of conflict probably arises from a lack of understanding as to the distinctions between physical and logical interfaces. The only application to application interface shown in the figures is a class 6 logical interface. This interface allows for the passage of data from one application to another logically. The only physical interface applications have is a class 5 interface to systems software. This paragraph will be expanded to specifically define that "No application to application physical interface is allowed".

**Paragraph 2.2.3.6**

"The SGOAA Model--". Example of my difficulty with terms. Also "The SGOAA shall provide -- sampling rate, data storage. --". Another example. An architecture does not involve those things.

**Response:**

Agree. The architecture must provide the interfaces for the implementation of these capabilities. Paragraphs 2.2.3.6, 2.2.3.7 and 2.2.3.8 will be modified to reflect this.

**Paragraph 2.2.3.8**

"the following." Something is missing.

**Response:**

Agree. The paragraph was rewritten to reflect that the standard interface consists of a command and control interface, a payload data communications interface and a payload event timing interface.

**Paragraph 2.2.4.1.3**

"SGOAA Model shall provide resources and interfaces --". Again, I think a clean definition of terms will tidy this kind of thing up.

**Response:**

Rewritten to attain consistency of terms and update to revised definitions of critical condition categories as defined in Table 2-1.

**Paragraph 2.2.4.1.4**

Perhaps the distinction intended between "configuration control" and "system configuration control" should be spelled out.

**Response:**

"Configuration Control" reference was deleted. I know of no difference between "configuration control" and "system configuration control". The entire paragraph was rewritten for clarity.

**Paragraph 2.2.4.1.9**

"without requiring shutdown of total SGOAA" - ????

**Response:**

Rewritten to state: "The SGOAA shall support avionics hardware reintegration (introduction of flight hardware previously removed from operation) without requiring shutdown of the total avionics system"



**Paragraph 2.2.4.2.6**

reliability of the architecture model shall achieve a MTBF etc ??

**Response:**

**Rewritten to state:** " SGOAA compliant systems shall achieve the Mean Time Between Critical Failure (MTBCF) and Mean Time Between Failure (MTBF) as listed in Table 2-5."

**Paragraph 2.3**

I found this section very helpful reading. Will there be any identification of common elements in the next version? I still have the sense of a collection of autonomous subsystems; is that intended? JIAWG has a strategy to define "functionally interchangeable and integrated avionics modules", albeit for things like weapon systems. They view the development of such systems as VHSIC-based (in part, because of their self-test and diagnostics capabilities and the 50-70 percent reduction in module count they expect from VHSIC circuits). Other technologies, such as machine intelligence and multiuse sensors, are expected to contribute to the implementation of a truly distributed architecture. Will your SGOAA support such implementations adequately? I do not currently have the depth of knowledge to say. You may want to study some of the JIAWG reports on their Advanced Avionics to determine if there is anything there for NASA to leverage.

**Response:**

1. The functional entities presented in section 3 for the Space Data System Services and the Space Operations Control System are "common elements".
2. The subsystems are not autonomous. Although each subsystem is a modular element that may or may not be required for a specific system the subsystems required for a specific application must communicate and support each other in accomplishing the system purpose.
3. The SGOAA is a functional service and interface architecture, not a technology based architecture. The SGOAA should be capable of being implemented in essentially any technology base.

**Figure 3-1**

Consider making this into 3 charts: A Functional Overview with the 2 major types of processing broken out into only the 11 boxes. Then add tables/figures for the

subsequent detail of these 11 boxes divided into Control Processing and Info Processing (only 5 and 6 boxes detailed, respectively)

**Response:**

This figure is not, nor is it meant to be, a definitive list of all space processes. It adequately conveys the information intended in its present form.

**Paragraph 3.2**

Europe's mandated hierarchical Object-oriented Design (HOOD) technique and notation might help with the "hybrid" approach described here.

**Response:**

The HOOD technique and notation will be investigated to determine the utility of use for this application.

**Figure 3-2**

I would suggest a redrawing of this which highlights (by position or elimination) the only two of the 8 bubbles you intend to describe. Alternatively, you could just illustrate the bubbles in the "normal mode" and have the text explain your focus.

**Response:**

The introduction to section 3.3 has been rewritten. Figure 3-2 has not been modified because it is intended to distinguish all primary (or typical usage of "avionics") from other (support) avionics which are important to successful operation, such as the data system or launch support elements. Only the support, core avionics are addressed by this section, as noted in the re-written introduction.

**Paragraph 4.2**

The recommendation in section 3.4 should be recaptured here or it may get lost.

**Response:**

The recommendation has been captured.

~~SAE~~ ✓ stet  
**C.3 SAE COMMENTS**

**C.3.1 CTA INC COMMENTS**

Feedback on the "SGOAA Standard Specification" as reviewed multiple times by CTA Inc.,  
D. Cooper, 28 Jan. - 4 March 1993.

**Comment 1.1:**

Suggest reformatting the document outline to more closely follow the other DOD standards  
such as MIL-STD-1553.

1. SCOPE
  - 1.1 Scope
  - 1.2 Application
2. REFERENCED DOCUMENTS
  - 2.1 Standards
  - 2.2 Other Documents
3. DEFINITIONS
4. GENERAL REQUIREMENTS
5. NOTES

**Response:**

Concur, the specification has been completely re-written to address this point and  
related structural comments made by Dave Cooper.

**Comment 1.2:**

Suggest rewording the "Scope" section to make it clear as to what you are trying to  
standardize. For example, the section should start with the words "This standard

establishes requirements for ...." It wasn't clear to me from reading the document in its present form just how to fill in the blank.

**Response:**

Concur.

**Comment 1.3:**

Most sections of the document seem to provide more information on "why" something is being suggested rather than specifying precisely "what" the requirements really are as they would pertain to the system designer.

**Response:**

The why's have been deleted (because they are in the companion technical guide) or where especially important to understanding, they have been moved to a new NOTES section in the specification.

**Comment 1.4:**

Requirements should be stated in such a way that they are obviously testable. For example, "Voltage on Pin A, connector P42, shall be between 4.5 to 5.5 VDC in the quiescent state." The majority of the requirements shown on pages 12-14 do not meet these criteria.

**Response:**

Agree, however, this is a generic specification intended to be tailored to specific missions and systems, which are not (at this time) known. It is not clear how to make specific quantitative requirements which are independent of specific missions and which can be tailored during system development. Perhaps some kind of formula might work here.

**Comment 1.5:**

The organization section of the document will not be necessary once the MIL-STD type format is used.

**Response:**

Agree, it has been deleted.

**Comment 1.6:**

Definitions should be shortened to specify only what a particular word or phrase means in the context of this document. Many of the current definitions seem to include application guidance or statements of rationale or clarification that should not be in this section.

**Response:**

Agree, they have been made more specific.

**Comment 1.7:**

There are inconsistencies in several of the definitions. For instance your definition for Application Platform comes right out of the POSIX P1003.0 definition section and contains a reference to an "application" or "application software". In this context, I expected your definition of "application" to also be consistent with POSIX. It is not. All definitions should be reviewed for consistency.

**Response:**

Concur, all definitions based on POSIX have been changed for consistency where needed and double checked against the P1003.0 document, and exceptions have been specifically noted.

**Comment 1.8:**

Titles of referenced documents such as are shown in Table 3.2 need to be accurate. For instance, MIL-STD-1553 is not an application handbook.

**Response:**

Agree, titles will be checked.

**Comment 1.9:**

The Architecture section needs to contain testable requirements. Listing six interface classes and not giving specific implementation guidance is ineffective. It is, however, a good way to breakdown the various types and levels of interface transactions that occur within a system, and it does provide a way to compartmentalize a system design. But to make a standard out of this document, a lot more detailed requirements need to be developed.

**Response:**

Agree that testable requirements are needed. However, as noted in response 1.4, this is a generic specification intended to be tailored to specific missions and systems, which are not (at this time) known. It is not clear how to make specific testable

requirements which are independent of specific missions and which can be tailored during system development. More detailed work is needed; that is why NASA has approached the SAE.

**Comment 1.10:**

The use of the term "physical interface" should be limited to describing mating surfaces between physical devices such as connector halves. This usually includes such things as connector type, shell size, pin insert arrangement, pin size, etc. Any other use of the term physical interface would be non-standard.

**Response:**

The use of the term "physical interface" has been changed to resolve this confusion. This term was being used in the sense of "direct interface", and the term has been changed to use the latter phrasing for clarity.

**Comment 1.11:**

I saw no rationale as to why this particular architecture was chosen to be your generic "standard". You are advocating a distributed processing environment with centralized control. This may not be appropriate for every future NASA requirement, and, therefore may be too restrictive.

**Response:**

There is no intention of restricting the architecture to centralized control. This is a misunderstanding due to unclear figures. The figures have been re-done to clarify that centralized or distributed processing and control are both acceptable to the architecture.

**Comment 1.12:**

I would have expected to see a requirement for a vehicle health management system in an architectural standard such as this. General guidance pertaining to the required levels of fault detection, isolation, and reporting, timing requirements for fault detection and reporting, required degraded operational capabilities, and redundancy requirements need to be addressed.

**Response:**

Vehicle health management is only one service needed in this architecture. There are others just as important (for instance an operating system). The specification has

been re-written to identify all the categories of services needed. Vehicle health management is one service under the data system manager service.

**Comment 2.0:**

**CONCLUSION:**

The document presently reads more like a study report than a standard. There are some good ideas presented concerning the interface classes, but the material needs to be presented in the form of testable requirements to be levied on system designers. The more general question is whether you want to try to standardize on a single architecture for all classes of applications, or whether it would be better to try to standardize on interface definition requirements and on certain functional requirements for fault isolation and reporting requirements regardless of the selected architecture.

**Response:**

The re-written standard should no longer read like a study report. This was originally done to provide the flavor of the POSIX documents. Agree (as noted above) on the desirability of testable requirements. A single architecture seems feasible if it is set up as the SGOAA is, that is with minimal constraints on actual implementation except for key interfaces and key services. Fault isolation (as important as it is) must not be allowed to outweigh other services just as important.

**Comment 3.0:**

**RECOMMENDATIONS:**

I would recommend that the following actions be taken before turning the document over to the SAE for consideration as a standard.

**Recommendation 1.**

Reformat the document to comply with general instructions for preparation of Government Standards.

**Response:**

Concur - has been done.

**Recommendation 2.**

Tailor the definitions contained in this document to this specific application.

**Response:**

Concur - has been done.

**Recommendation 3.**

Phrase each of the requirements in the document so that a method of compliance testing is implicit in the statement of the requirement.

**Response:**

Concur - have attempted to, but not clear how to implement this recommendation in many cases.

**Recommendation 4.**

Fill in the missing words for "this standard establishes requirements for ... "

**Response:**

Concur.

**Additional Comments On The SGOAA:**

**Comment 4:**

Doesn't state a problem, therefore the "solution" can only be judged on a presumed and arbitrary basis.

**Response:**

Do not agree, this is a standard for a solution, not a study of problems to be solved.

**Comment 5:**

Interpreting the "benefits" listed on page one seems to imply that the SGOAA is directed toward standard interfaces, commonality, and modular interchangeability of software and hardware. All of which are good and worthwhile *when carefully weighed and tailored to all the other specifics of a given mission problem*. However, none of the benefits listed are derived from the proposed "architecture". Rather these benefits are only derived from a comprehensive set of *module* standards that are by definition independent of the architectural context.

**Response:**

The standard has been re-written to clarify the use of standard interfaces and standard services with pre-defined interfaces. The purpose of these standards is to provide standardized and common elements which would act as requirements to be



used in systems design just as mission requirements have to be used in systems design. Thus these standard requirements would be tailored to specific mission and system needs. Module standards would be just one of many lower level standards needed in conjunction with this standard, since the SGOAA is intended to be an *umbrella* standard.

**Comment 6:**

The paper provides no analysis or discussion of alternative architectures, nor any justification for the one presented; hence, it would be pure conjecture to attempt, to establish the soundness of the SGOAA chosen. More importantly the real purpose of the SGOAA seems to be to help in the process of identifying an appropriate set of interfaces that need to be addressed in order to ensure that all required *module* interface standards are driven out.

**Response:**

Again, this is a specification for a solution to be used in tailoring to specific mission and system needs. No alternatives were discussed because they are not relevant. Standard interfaces and standard services are the key requirements established in the SGOAA. Module interfaces (for both hardware and software) are only one kind of interface being identified as needing to be standardized.

**Comment 7:**

The standard totally overlooks the many critical interfaces in the power, thermal, structural, electrical shielding/grounding, etc. areas.

**Response:**

No, it does not overlook such critical interfaces. These are all physical interfaces within the Class 1 interface. They would be defined in lower level standards pointed to by the SGOAA parent standard.

**SUGGESTIONS:**

**Recommendation 5:**

The present title of this paper implies that a standard architecture is being proposed when that is not only not necessary for the stated benefits but is actually not desirable. With standard modules as building blocks all of the proposed benefits can be achieved without limiting the architecture in any way. To limit the avionics architecture to a "standard"

would remove the most powerful tool the system designer has, namely: the ability to match the system architecture to the problem "architecture".

**Response:**

This is a misunderstanding of the usage of the phrase "architecture". In the SGOAA, we are using architecture to mean the overall structure and organization of software, hardware and interfaces (both internal and external) needed to make a system operate. Agree that standard modules are needed, but they have to be carefully identified and their interfaces clearly defined. That is the purpose of SGOAA – to identify the key interfaces and services needed in a standard approach. Not to specify that one centralized or another distributed processing architecture is the best one. There is no undue limitation on designers – they are only constrained from re-inventing the "wheel" just because they want their own wheel and not someone else's wheel.

**Recommendation 6:**

Consideration needs to be given to all the neglected interface areas mentioned above since these are at least of equal significance for space applications and may in fact be much more difficult to establish since they are unique and have no commercial counterparts.

**Response:**

Agree, no interfaces should be neglected. We believe the SGOAA does not neglect any interfaces.

**APPENDIX D**  
**LIST OF REFERENCES**



## **D. LIST OF REFERENCES**

- [BOE91] Flanagan, Rich and Van Ausdal, Art, "SATWG Flight Data System Architecture Specification Outline" briefing, 25 October 1991
- [GD90A] General Dynamics "Space Avionics Requirements Study", 21 October 1990 as briefed to the SATWG
- [ISO7498] "Information Processing Systems - Open Systems Interconnection - Basic Reference Model", International Standards Organization, 1984.
- [JSC 31000] Space Station Projects Description and Requirements Document, Vol 3, Rev G, 4 April 1991.
- [POSIX91] "Draft Guide to the POSIX Open Systems Environment", P1003.0/D14, IEEE Computer Society, November 1991.
- [PRU90] Pruett, D., "Avionics Software Open System Environment Reference Model", JSC, March 1990.
- [WRA91] Wray, R.B., "Requirements Analysis Notebook for the Flight Data Systems Definition in the Real-time Systems Engineering Laboratory (RSEL)," Job Order 60-430 for the JSC, NASA-CR-185698, LESC-29702, December 1991.
- [WRA93] Wray, R.B. and Stovall, J.R., "Space Generic Open Avionics Architecture (SGOAA) Reference Model Standard Specification", Job Order 60-430, Contract NAS 9-17900 for the JSC, NASA-CR-188245, LESC-30354-A, March 1993.



**DISTRIBUTION LIST FOR LESC-30347-A  
SPACE GENERIC OPEN AVIONICS ARCHITECTURE  
(SGOAA) STANDARD**

**NASA**

EK111/D. M. PRUETT (5)  
PT4/E. M. FRIDGE (5)  
AMES/E. S. CHEVERS (10)  
~~JM-2/S. McDONALD (12)~~

EG1/D. P. BROWN  
EG111/K. J. COX  
JPL/MS 301-235/A. HOOKE  
IA13/D. STONE

**LESC**

C18/J. R. THRASHER  
C18/E. A. STREET  
C18/R. E. SCHINDELER  
C18/G. Y. ROSET  
C18/J. STOVALL (10)  
C106/P. G. O'NEIL  
C07/J. E. MOORE

C18/G. L. CLOUETTE  
C18/R. W. WRAY (10)  
C18/M. W. WALRATH  
C18/B. L. DOECKEL  
B11/G. J. MOORMAN  
C83/S. J. THOMAS  
C22/D. CRAVEY  
C29/P. HOPKINS

C87/M. W. BRADWAY  
(FOR SATWG)

C18/JEAN FOWLER  
(MASTER + 2 COPIES)

B16/LESC LIBRARY (2)

**SAE/ASD**, SAE INTERNATIONAL, 400 COMMONWEALTH AVE, WARRENDALE,  
PA. 15096  
BARBARA ROTH (FILE)  
JOHN MEYER

**MITRE**, 1120 NASA ROAD 1, HOUSTON, TX 77058  
D. ERB

**UHCL**, UNIVERSITY OF HOUSTON - CLEAR LAKE, 2700 AY AREA BLVE. -  
BOX 444, HOUSTON, TEXAS 77058  
CHARLES HARDWICH

**NIST/CSL**, FRITZ SCHULTZ, BLDG 225, ROOM B266, GATHISBURG, MD. 20899

**ROME LABS/OCTS**, GRIFFIS AFB, NY 13441-5700  
RICHARD WOOD

**DISTRIBUTION LIST FOR LESC-30347-A  
SGOAA STANDARD - CONTINUED**

**LASC**, 86 S. COBB ST., MARRIETTA, GA. 30063  
RICK HARWELL, D/73-D2, ZONE 0685  
JOHN WEAVER, D/73-D1, ZONE 0685  
COX, JIM, D/73-MA ZONE 0081  
REED, MIKE, D/73-MA, ZONE 0081  
HUDSON, ROCKY, D/73-D2, ZONE 0685

**LMSC**, 1111 LOCKHEED WAY, SUNNYVALE, CA. 94088-3504  
CHARLES TADJERAN, ORG. 62-31, BLDG 150  
ROY PETIS, ORG. 78-10, BLDG 264  
JOHN McMORRIS, ORG. 81-90, BLDG 157  
DUWAYNE DICKSON, ORG. 80-06, BLDG 154  
F. L. (FRED) LORY, ORG. 68-15, BLDG 104  
MERLIN DORFMAN, ORG. 62-80, BLDG 563

**LMSC (RD&D)**, 3251 HANOVER STREET, PALO ALTO, CA 94303-1191  
BILL GUYTON, ORG. 92-20, BLDG 254E  
RAY MUZZY, ORG. 90-21, BLDG. 254E  
STEVE SHERMAN, ORG. 96-10, BLDG 254E

**LOCKHEED-SANDERS**, 95 CANAL ST. NASHUA, NH 03061  
RAY GARBOS (NAM5D-5002)                      JEFF E. SMITH (PTP2-B002)  
JOHN MILLER (NCA 09-1106)                      DUNCAN MOORE (MER 24)

**LADC**, P. O. BOX 250, SUNLAND, CA. 91041  
ALEX LOEWENTHAL, DEPT. 25-14, BLDG 311

**LAD**, P. O. BOX 17100, AUSTIN, TX. 78744-1016  
CURTIS WELLS, ORG. T2-10, BLDG 30F

**LOCKHEED CORP.**, 4500 PARK GRANADA BLVD, CALABASIS, CA 91399-0310  
MICHAEL CARROLL  
BART KRAWETZ

**LAS Ontario**, P. O. BOX 33, ONTARIO, CA. 91761-0033  
C. R. (BOB) FENTON

**FWC**, P. O. BOX 748, FORT WORTH, TX 76101  
PAUL DANIEL, MAIL ZONE 2640

**LSOC**, 1100 LOCKHEED WAY, TITUSVILLE, FL 32780  
L. J. (LEWIS) BOYD, ORG. 32-40, (Z/LSO-183)  
ARTHUR EDWARDS, ORG. 11-42, BLDG. B/DX-D, Z/LSO-284)



**DISTRIBUTION LIST FOR LESC-30347-A**  
**SGOAA STANDARD - CONTINUED**

**BOEING CORP.** PO BOX 3999, SEATTLE, WA 98124-2499  
RICHARD FLANAGAN  
AL COSGROVE

**COMPUTING DEVICES INTL.** 8800 QUEEN AVENUE SOUTH, BLOOMINGTON,  
MN 55431  
JIM JAMES, M/S BLCN2A  
DOCK ALLEN

**WESTAR CORP.** 6808 ACADEMY PKWY EAST, NE, BLDG C, SUITE 3,  
ALBUQUERQUE, NM 87109  
CHRIS DE LONG

**HG USAF/SCS.** 1250 AIR FORCE, PENTAGON, WASHINGTON, D. C. 20330-1250  
COL ROBERT HANLON

**ROCKWELL INT'L CORP.,** 12214 LAKEWOOD BLVD., DOWNEY, CA. 90241  
SUMI MATSURA

**TRW,** HOUSTON, TX 77058  
DOUG RUE (NASA MAIL)

**FAIRCHILD SPACE,** 20301 CENTURY BLVD., GERMANTOWN, MD. 20874  
JOHN SCHNEIDER, FLIGHT DATA SYSTEMS

**E-SYSTEMS,** P. O. BOX 12248, ST. PETERSBURG, FL. 33733-2248  
JIM BRADY/MS29

**EER SYSTEMS INC.,** 3027 MARINA BAY DR., SUITE 105,  
LEAGUE CITY, TX 77573  
RAY HARTENSTEIN

**ROCKWELL INTL CORP.-ROCKETDYNE DIV.,** 6633 CANOGA AVE, P. O. BOX  
7922, CANOGA PARK, CA. 91309-7922  
ANTHONY THOMPSON, D1055-LB33

**RESEARCH ANALYSIS AND MAINTENANCE INC.,** 512 AUDUBON ST.,  
LEAGUE CITY, TX 77573  
ROGER EVANS

**M&AE,** 1200 G. STREET, NW, SUITE 800, WASHINGTON DC, 20005  
JOHN KELLER

**DISTRIBUTION LIST FOR LESC-30347-A**  
**SGOAA STANDARD - CONCLUDED**

**C.S. DRAPER LABS**, 555 TECHNOLOGY SQUARE, CAMBRIDGE, MA 02139  
J. BARTON DEWOLFE/MS 61

**SBS ENGINEERING**, 5550 MIDWAY PARK PLACE, NE, ALBUQUERQUE, NM  
87109  
MR. DEREK HEAD

**NAVMAR APPLIED SCIENCES CORP.** 65 WEST STREET, SUITE C200,  
WARMINSTER, PA 18974  
MR. DOUG D'AVINO

**MR. MARTIN FREED**, (ASC/ENASC), 5565 BARBANNA LANE, DAYTON, OH  
45415

**ASC/YFMXT**, WRIGHT-PATTERSON AFB, OH 45433  
MR. BYRON STEPHENS

**NAVAL AIR WARFARE CENTER**, AIRCRAFT DIVISION, WARMINSTER, PA  
18974-0591  
RICHARD J. PARISEAU/CODE 102A  
RICHARD S. MEJZAK/CODE 2021

**TEXAS INSTRUMENTS**, 6550 CHASE OAKS BLVD, PO BOX 869305, PLANO, TX  
75086  
DR. CHUCK ROARK/MS 8481

**HONEYWELL INC.**, 3660 TECHNOLOGY DR, MINNEAPOLIS, MN 55418  
MR RON FRAZZINI

**PARAMAX SYSTEMS CORP.** PO BOX 64525, ST PAUL, MN 55164-0525  
MR DARYLE HAMLIN/MS U1F15

**CTA INC.**, SUITE 310, 18333 EGRET BAY BLVD, HOUSTON, TX 77058  
MR DAVID COOPER

**MITRE CORPORATION**, 202 BURLINGTON ROAD, BEDFORD, MA 01730-1420  
WILLIAM T. BRANDOM/D-96  
JACK SHAY/DIRECTOR OF SYSTEMS DEVELOPMENT

**MR ED SMITH**, EXECUTIVE VICE PRESIDENT, NCOSE,  
1907 BELLMEADE, HOUSTON, TX 77019